Self checking circuits applied to the fault diagnosis of microprocessor based systems

Hollis, T. J.

Award date:
1986

Awarding institution:
University of Bath

SELF CHECKING CIRCUITS APPLIED TO THE FAULT DIAGNOSIS OF MICROPROCESSOR BASED SYSTEMS

submitted by T. J. Hollis
for the degree of PhD
of the University of Bath
1986

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

This thesis examines the fault diagnosis of microprocessor based systems from the use of existing test techniques and equipment to the design and application of self checking circuits.

Initially, some existing techniques and equipment for the testing of microprocessor based systems are detailed, together with the test philosophies of twelve major manufacturers or users of such systems. This includes a specific application of in-circuit emulation.

Consideration of the testing process at the design stage has resulted in built in test at board and chip level. Common forms of built in test are studied, particularly error detection codes, and a summary provided of available integrated circuits with built in test.

The aspect of built in test where a system is tested concurrently with normal operation makes extensive use of fault detection or checking circuits. The problem of failures in these checking circuits is resolved with the use of self checking circuits.

The theory of self checking circuits is introduced with formal definitions of their characteristics. Based on the tests required to detect all single failures in logic gates and the use of Karnaugh maps, a technique is proposed for the design of totally self checking circuits. Circuit designs are presented for several self checking code checkers.

The mechanisms required to construct a self checking microprocessor based system are discussed. These allow a device or circuit failure in the system to be precisely located and include the use of signal isolation circuits. A review of proposals for self checking devices and systems reveals how extensively the various self checking mechanisms are adopted.

Finally, an experimental self checking computer is described, in which the application of self checking circuits to the fault diagnosis of microprocessor based systems is practically evaluated.

ACKNOWLEDGEMENTS

At the University of Bath, the author wishes to thank Mr. A. R. Daniels (Senior Lecturer in the School of Electrical Engineering) for his guidance as Supervisor during the research and for the provision of facilities within the School of Electrical Engineering. The guidance given Dr. S. L. Hurst (Senior Lecturer in the School of Electrical Engineering) is also gratefully acknowledged. In addition, the author appreciates the assistance given by Dr. R. T. Lipczynski (Lecturer in the School of Electrical Engineering) during the industrial survey.

The provision of a Case Award by the Science and Engineering Research Council, Swindon, is also acknowledged.

At Rolls Royce plc, Bristol, the author wishes to thank Mr. R. D. Fyfe (ADR Applications Manager) and Mr. A. C. Plenty (Head of Electronics and Instrumentation) for the provision of time and facilities during the past three years, as well as Mr. P. L. Westlake (Systems Planner) for many helpful discussions and Mr. M. S. Whitfield (ADR3 Project Manager : Systems Procurement) as industrial supervisor during the first three years.

The assistance given by Ms. H. M. R. Hollis (Architectural Designer at the Welsh School of Architecture, Cardiff) and Mr. A. Burn (Senior Systems Engineer at Rolls Royce plc, Bristol) in the preparation of figures for this thesis is gratefully acknowledged, as is the assistance given by Mr. P. D. Burn (Sales Engineer at Multibloc Sales Ltd., Keynsham, Bristol) in the wiring of the experimental computer.

During the industrial survey, the co-operation of the following companies is noted; Lucas Aerospace Ltd., Hemel Hempstead; Satchwell Control Systems Ltd., Slough; British Aerospace, Dynamics Group, Stevenage; Renishaw Electrical Ltd., Wotton-Under-Edge; Marconi Avionics Ltd., Microprocessor Division, Milton Keynes; Digital Equipment Company Ltd., Bristol; Ferranti Computing Systems Ltd., Bracknell; Data General, Bristol; Cifer Systems Ltd., Melksham; Negretti and Zambra Avionics Ltd., Eastleigh; and Square 'D' Ltd., Swindon.

Finally, the author is indebted to his parents, Mr. and Mrs. R. J. Hollis, for their constant support and encouragement.

SYMBOLS AND ABBREVIATIONS

A	Address line
ACIA	Asynchronous Communications Interface Adaptor
ALU	Arithmetic Logic Unit
ASCII	American Standard Code for Information Interchange
ATE	Automatic Test Equipment
B	Buffer
BAUD	bits per second
BIT	Built In Test
C	Checker
CK,CLK	Clock
CLR	Clear
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
d	Hamming distance
D	Diode or Data line
DC	Direct Current
DED/DEC	Double Error Detecting/Double Error Correcting
DIR	Direction
DMA	Direct Memory Access
EXOR	Exclusive-OR
F_s	secure fault set
F_t	tested fault set
G	enable
ECL	Emitter-Coupled Logic
EPROM	Erasable Programmable Read Only Memory
Hex	Hexadecimal (base 16)
Hz	Hertz
I	Isolator
I/O	Input/Output
IC	Integrated Circuit
ICE	In-Circuit Emulation
I^2L	Integrated Injection Logic
k	kilo (10^3)
K	1024
LA	Logic Analayser
LED	Light Emitting Diode
LFSR	Linear Feedback Shift Register
LSI	Large Scale Integration
M	Mega (10^6)
MOS	Metal-Oxide Semiconductor
MSI	Medium Scale Integration
MUX	Multiplexer
N	secure input set
PCI	Programmable Communications Interface
PSC	Partially Self Checking
PIA	Peripheral Interface Adaptor
PLA	Programmable Logic Array
PR	Preset
Q	flip-flop output
R	Resistor
RC	Resistor Capacitor
R/$\overline{W}$	Read/$\overline{\text{Write}}$

RAM	Random Access Memory
ROM	Read Only Memory
RST	Reset
SA	Signature Analysis
SA0	Stuck-At-0
SA1	Stuck-At-1
SC	Self Checking
SED/SEC	Single Error Detecting/Single Error Correcting
SSI	Small Scale Integration
ST0	Self Testing Only
SV	Self Verification
TPG	Test Pattern Generation
TR	Transistor
TSC	Totally Self Checking
TTL	Transistor-Transistor Logic
U	the Universal set
µP	microprocessor
V	Volts
V_{cc}	Positive supply rail
VDU	Visual Display Unit (terminal)
VLSI	Very Large Scale Integration
VMA	Valid Memory Address
X	normal input set
X_i	vector X
x_i	variable x within a vector
20mA RS232	interface specifications
=	equals
≠	does not equal
≏	approximately equals
≡	is equivalent to
<	less than
≤	less than or equal to
>	greater than
≥	greater than or equal to
< >	vector
+	addition, or logical OR
$+_N$	addition modulo N
Σ	summation
⊕	Exclusive-OR
.	logical AND
!	factorial
U	the Union of
∈	is a member of
∉	is not a member of

$\Rightarrow$	implies that
$\supset$	is a superset of
$\subset$	is a subset of
$\subseteq$	is a subset of or equal to
$\forall$	for all
$\nexists$	for not all
$\mid$	given that
$\exists$	there is some
$\rightarrow$	to or maps to
$\leftarrow$	from
$\{ \}$	Set
ϕ	clock phase or null set
$\lceil \rceil$	rounded up to the nearest integer
$\lfloor \rfloor$	rounded down to the nearest integer
$\&_M$	Morphic AND
$\bar{A}$	logical complement of A or A is active low
①, ⓪	codewords on a Karnaugh map

CONTENTS

CHAPTER ONE : INTRODUCTION

A continuing increase in the complexity of integrated circuits, namely Very Large Scale Integration (VLSI), has meant more logic power per square millimetre of circuit area.

The introduction of the microprocessor has led to a major advance in product design, with a reduction in the design time and a realisation of products which were previously inconceivable.

However, this increased density of integration has resulted in internal circuits or modules becoming transparent to the outside world, which means that they cannot be directly controlled or observed. This is certainly true of microprocessors, which have the added complexity of a bus structure, so that it is necessary to observe the activity on the address data and control buses simultaneously to determine its exact operation at any time.

These factors have contributed to the creation of a serious challenge to the electronics industry; namely how to service processor boards and the products which use them. Testing and troubleshooting using conventional equipment is a difficult and time consuming task, which requires skilled personnel, of which there is generally a shortage, since most of them are actively involved with design.

In order to combat this problem, the industry has been searching for new tools, techniques and equipment which can be used in the field, at a central or local repair base and also on the production line, by people who are not necessarily familiar with the operation and structure of such systems. Design and debugging during development also benefit from these techniques.

Some general constraints for test techniques or equipment

are:

1) An ability to work with existing and future processors.
2) Provision for operating with various logic types.
3) Facilities for testing the analogue sections of a system.
4) Speed for real time measurement.
5) Ease of installation into the unit under test.

Expanding these points:

1) In order not to become rapidly outdated with the introduction of new processors, the equipment must be able to work with a wide variety of processors, supporting eight and sixteen bit architectures, and be able to cater for new ones as they become available, notably those with thirty-two bit architectures. This is generally achieved by using a dedicated probe for each processor and sufficient flexibility in the main instrument to cope with present and future needs.

2) The use of a wide variety of technologies, notably within interface circuits, has resulted in a requirement to operate with different voltage levels, whether these be those of TTL, MOS or ECL.

3) Since analogue/digital interfaces are crucial to the diversity of processor based systems, it is an advantage if analogue testing can be handled as well. Much analogue testing now relies on the digitisation of its signals for subsequent processing and display.

4) The speed at which the test equipment operates should be higher than that of the system it is testing. This means that the stimulation and observation of reponses can be carried out in real time. Problems due to timing and crosstalk, for example, are less evident at slower speeds.

5) A lot of test equipment has been designed around the IEEE488 bus [1.1], which allows a rapid connection of the instrument to the unit under test, a defined communication link, as well as compatibility with other instrumentation.

Much has been written on current techniques and equipment, but to get a practical viewpoint, a survey of twelve major manufacturers and/or users of microprocesser based systems was instigated to establish their particular test philosophies. The various strategies adopted by these companies are detailed in Chapter 2 from the viewpoint of:

1) The benefits they offer.
2) Limitations to their performance.
3) The extent of their usage.

Considerable personal practical experience, obtained over a number of years, is also incoporated into the appropriate section. In addition, Appendix A describes a specific application of in-circuit emulation to a microprocessor based system.

Most of the techniques in extensive current use require some form of external equipment to perform the testing. With technology progressing so rapidly, this equipment soon becomes inflexible or is simply unable to cope. Testing can become a costly process if the equipment has to be replaced.

During the past twenty years, considerable thought has been given to catering for the test process at the design stage and not as an after-thought when the design, manufacture and installation phases of a product have been completed. This is Built-In-Test (BIT), which is introduced in Chapter 3 with a discussion of faults and fault models, observability and controllability, design for testability and redundancy. These four concepts are used

to describe and specify BIT. In Chapter 4 the more common forms of BIT are studied, again considering the benefits and limitations of each technique. The use of error detecting codes, scan design, built in logic block observers, autonomous testing, serial shadow registers and design for self verification are included in this study.

Figure 1.1 summarises the possible test strategies for VLSI (equipment, techniques and built-in hardware or software) and details many references for further information.

An integral part of many of the BIT strategies discussed in Chapter 4 is an ability to check a microprocessor based system whilst it is on line carrying out its normal functions. However, what happens if one or more of the checking circuits fail? Is there a checker checking the checker? The solution is to use self checking circuits, circuits which effectively check themselves. The rest of the thesis is devoted to this subject.

In Chapter 5 self checking circuits are introduced with formal definitions of their characteristics. This includes the concepts of self testing only (STO) circuits, partially self checking (PSC) circuits and totally self checking (TSC) circuits. Self checking (SC) networks, SC checkers, SC bit-sliced circuits, SC byte-sliced circuits and SC sequential circuits are also discussed.

Chapter 6 investigates the theory and design of SC circuits. The tests which have to be performed on individual logic gates to detect failures within them are significant in determining if a circuit is SC or not, so these tests are considered in some detail. Based on the resulting test requirements and the use of Karnaugh maps, a technique is then proposed for the design of SC circuits. Subsequent sections in Chapter 6 describe the design of TSC comparators and periodic signal checkers, as well as TSC 1-out-of-n and parity code checkers.

Since this thesis is concerned with the testing of microprocessor based systems, Chapter 7 discusses the requirements to make such systems self checking. The main objective here is a means of precisely locating a faulty device or circuit during normal operation. Various techniques are presented for fault detection and fault diagnosis. Chapter 8 then examines the extent to which these techniques are employed in current and past research by reviewing a number of self checking devices and systems.

In order to support the philosphies of Chapter 7 and the conclusions from Chapter 8, a minimal self checking microprocessor based system has been built using the designs of Chapter 6. This is described in Chapter 9 together with results from its subsequent evaluation. An essential requirement for the construction of this system was a means of fault isolation between the transmitting and receiving points of an interconnection; notably the address, data and control buses. This design work for unidirectional and bidirectional isolators is detailed in Appendix B.

Finally, overall conclusions on the design and use of SC circuits in microprocessor systems are presented, as well as suggestions for further work.

1.1 : REFERENCES

1.1) IEEE-488 instruments - A. Santoni; EDN; October 28, 1981; pp. 77-90.

1.2) How to troubleshoot and repair microcomputers - J. D. Lenk; Reston Publishing Inc.; Virginia, USA; 1980; pp. 47-61.

1.3) Troubleshooting microprocessors and digital logic - R. L. Goodman; Tab Books Inc.; PA., USA; 1980; Chapter 5.

1.4) Practical troubleshooting techniques for microprocessor systems - J. W. Coffron; Prentice Hall Inc.; New Jersey, USA; 1981; pp. 134-7.

1.5) as per 1.3); Chapter 4.

1.6) Digital testing oscilloscope cushions against mounting costs of troubleshooting - W. E. Shoemaker, D. W. Wilson; Electroncs; March 16, 1978; pp. 105-112.

1.7) ICE development heats up to handle new processors - D. G. West, R. Paulson; Electronic Design; September 3, 1981; pp. 125-130.

1.8) Fast emulator debugs 8085-based microcomputers in real time - M. Y. Yen; Electronics; July 21, 1977; pp. 108-112.

1.9) Hardware emulation conquers the testing mountain created by 16 bit uPs - D. Lemke, D. Smith, T. Tunder; Electronic Design;No. 14, July 5, 1979; pp. 54-59.

1.10) Cut hardware, software development costs - take advantage of in-circuit emulation - J. M. Kelley; Electronic Design; No. 17, August 16, 1979; pp. 66-71.

1.11) Use of the slave emulator - T. McCann; Microprocessors & Microsystems; Vol. 5, No.2, March 1981; pp. 65-68.

1.12) ROM simulator, memory tracer debug microsystems - J. L. Nichols, R. W. Ulrickson; Electronic Design; July 23, 1981; pp. 147-154.

1.13) as per 1.4); Chapters 3, 4, 5 and 9.

1.14) Emulator solves system-level debug problems - R. Parks, K. Greenberg; Electronic Design; May 14, 1981; pp. 173-180.

1.15) Fault diagnostics for microprocessor based systems - T. J. Hollis; Project report for the degree of

B.Sc. in Electrical Engineering; University of Bath, England; 1980; p. 16.

1.16) as per 1.2); pp. 273-284.

1.17) as per 1.4); Chapter 7.

1.18) Instruments - A. Santoni; EDN; July 20,1979; p. 97.

1.19) as per 1.15); pp. 13-15.

1.20) Signature analysis wins new acclaims - M. Marshall; Electronics;February 14, 1980; pp. 103-104.

1.21) Applying signature analysis to existing processor-based products - R. Rhodes Burke; Electronics; February 24, 1981; pp. 127-133.

1.22) as per 1.2); pp. 63-105.

1.23) Logic-analyzer 'computer' simplifies computer testing - G. Brock; Electronic Design; June 25, 1981; pp. 101-108.

1.24) What to look for in logic timing analysers - M. Marshall; Electronics; March 29, 1979; pp. 109-114.

1.25) Designing a servicemans' needs into microprocessor-based systems - M. Neil, R. Goodner; Electronics; March 1, 1979; pp. 122-128.

1.26) as per 1.4); Chapter 6.

1.27) Flexible pattern generation aids logic analysis - S. Palmquist, M. Pettet; Electronic Design; October 15, 1981; pp. 165-174.

1.28) Functional and in-circuit testing team up to tackle VLSI in the '80s - P. Hansen; Electronics; April 21, 1981; pp. 189-195.

1.29) Application of guided probe and in-circuit test architectures to compensate for inadequate UUT testability - A. Lowenstein; Autotestcon '80*; pp. 60-66.

1.30) Optimised digital testing using general purpose ATE - P. J. Hand; Autotestcon '80*; pp. 259-263.

1.31) ATE swings towards merged in-circuit, functional tests - J. McLeod; Electronic Design; October 29, 1981; pp. 90-104.

1.32) To sort out card testers, ask the right questions - S. L. Black; Electronic Design; February 5, 1981; pp. 115-119.

1.33) Automating test generation closes the design loop

- R. Hickling, G. Case; Electronics; November 30, 1981; p.129-133.

1.34) Test LSI boards functionally on an in-circuit tester - T. Jackson, P. Vais; Electronic Design; October 29, 1981; pp. 137-144.

1.35) Design for in-situ chip testing with a compact tester - C. C. Perkins, S. Sangani, R. Stopper, M. Valitski; 1980 Test Conf.*; pp. 29-41.

1.36) Software package generates in-circuit programs automatically - C. Bostrom; Electronic Design; February 5, 1981; pp. 99-102.

1.37) In-circuit tester takes on ECL, TTL, and MOS devices - M. P. Carroll, G.W. Peterson; Electronic Design; May 28, 1981; pp. 91-97.

1.38) Test vectors development and optimisation for a microprocessor - C. C. Timoc, L. M. Hess, F. R. Stott; Autotestcon '80*; pp. 165-170.

1.39) Interactive design simplifies test-program generation - B. Kelley, M. Bonham, R. Chruscial; Electronic Design; February 19, 1981; pp. 169-173.

1.40) Microprocessor functional testing - R. Chantal, S. Garbriele; 1980 Test Conf.*; pp. 433-443.

1.41) Enhanced simulator takes on bus structured logic - H. Levin; Electronic Design; October 29,1981; pp. 153-157.

1.42) as per 1.5); Chapter 10.

1.43) Reliability Design Handbook - R. T. Anderson; IIT Research Institute (IITRI), Chicago, for Reliability Analysis Centre; 1975; pp. 258-271.

1.44) as per 1.4); Chapter 8.

1.45) Testing logic networks and designing for testability - T. W. Williams, K. P. Parker; Computer; October 1979; pp. 9-21.

1.46) Design for autonomous test - M. J. McCluskey, S. Bozorgui-Nesbat; 1980 Test Conf.*; pp. 15-21.

1.47) Microprocessor bus standard could cure designers' woes - G. Force; Electronics; July 20, 1978; pp. 113-118.

1.48) A monolithic self-checking error detection processor - J. Chavade, M. Vergniault, P. Rousseau, Y. Crouzet, C. Landrault; 1980 Test Conf.*; pp. 279-286.

1.49) Quality and control self-test - R. W. Spearman, F. D. Patch; 1980 Test Conf.*; pp. 257-260.

1.50) Implementation techniques for self-verification - R. M. Sedmak; 1980 Test Conf.*; pp. 267-278.

1.51) BIDCO, Built-in digital circuit observer - P. P. Fasang; 1980 Test Conf.*; pp. 261-266.

1.52) Enhance computer fault isolation with a history memory - G. L. Fitzgerald; Autotestcon '80*; pp. 267-278.

1.53) Application of shift register and its effective implementation - M. Kawai, S. Funatsu, A. Yamada; 1980 Test Conf.*; pp. 22-25.

1.54) Incomplete scan path with an automatic test generation methodology - E. Trischler; 1980 Test Conf.*; pp. 153-162.

1.55) Applying the Hamming code to microprocessor-based systems - E. L. Wall; Electronics; November 22, 1979; pp. 103-110.

1.56) A research oriented microcomputer with built in auto-diagnostics - J. Moreira de Souza, E. Peixoto Paz, C. Landrault; FTCS-6*; pp. 3-8.

1.57) Self testing computers - J. B. Clary, R. A. Sacane; Computer; October 1979; pp. 49-59.

1.58) as per 1.43); pp. 185-213.

1.59) Analysis of fault detection coverage of a self-test software program - V. Tasar; FTCS-8*; pp. 65-71.

1.60) Design self-testing capability for reliable uC-system operation - S. Strom; EDN; October 28, 1981; pp.102-108.

1.61) Efficient and effective uC testing requires careful preplanning - E. S. Donn, M. D. Lippman; EDN; February 20, 1979; pp.97-107.

1.62) Design forethought promotes easier testing of microcomputer boards - M. D. Lippman, E. S. Donn; Electronics; January 18, 1979; pp.113-9.

1.63) Microcomputer for emulation bares hidden busses, functions - J. Moon; Electronics; July 17, 1980; pp. 126-129.

1.64) Self-testing supercells; Alternative test strategies - D. K. Bhavsar; Autotestcon '80*; pp. 135-139.

1.65) Off line, built-in test techniques for VLSI cir-

cuits - M. G. Buchier, M. W. Sievers; Computer; June 1982; pp. 69-82.

1.66) Design of self-checking MOS-LSI circuits, application to a four-bit microprocessor - Y. Crouzet, C. Landrault; FTCS-9*; pp. 189-192.

1.67) Memory finds and fixes errors to raise reliability of microcomputer - A. Heimlich, J. Korelitz; Electronics; January 3, 1980; pp. 168-172.

* see section B.5.

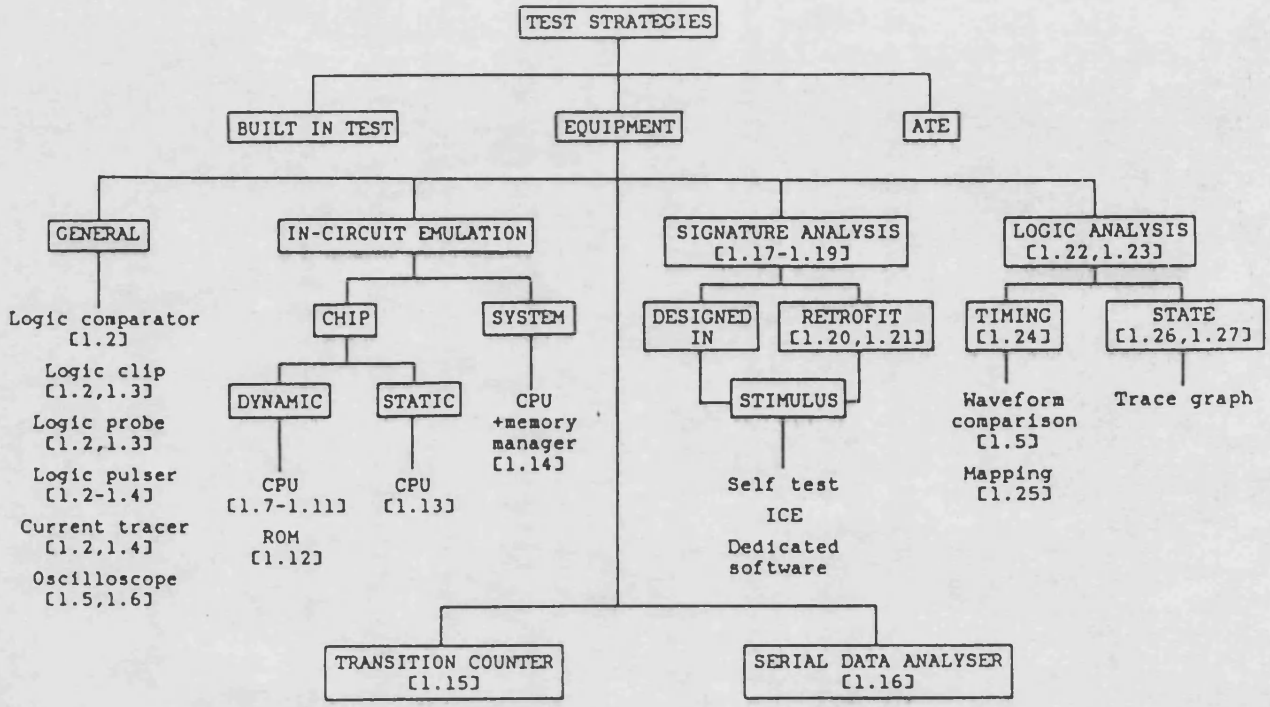


FIGURE 1.1(a) TEST STRATEGIES

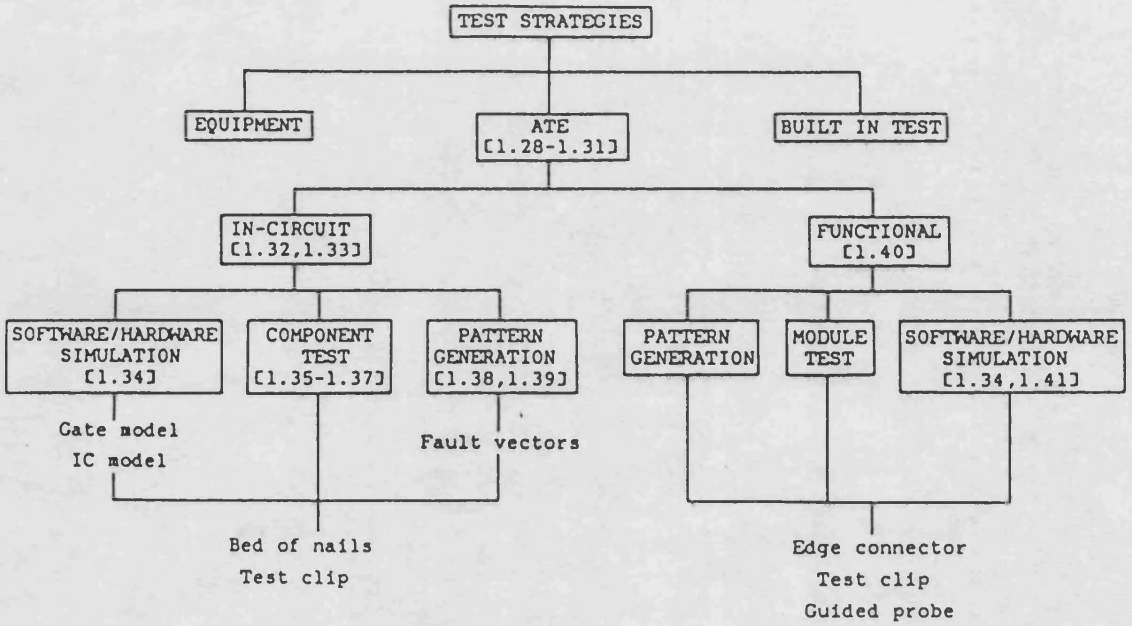


FIGURE 1.1(b) TEST STRATEGIES

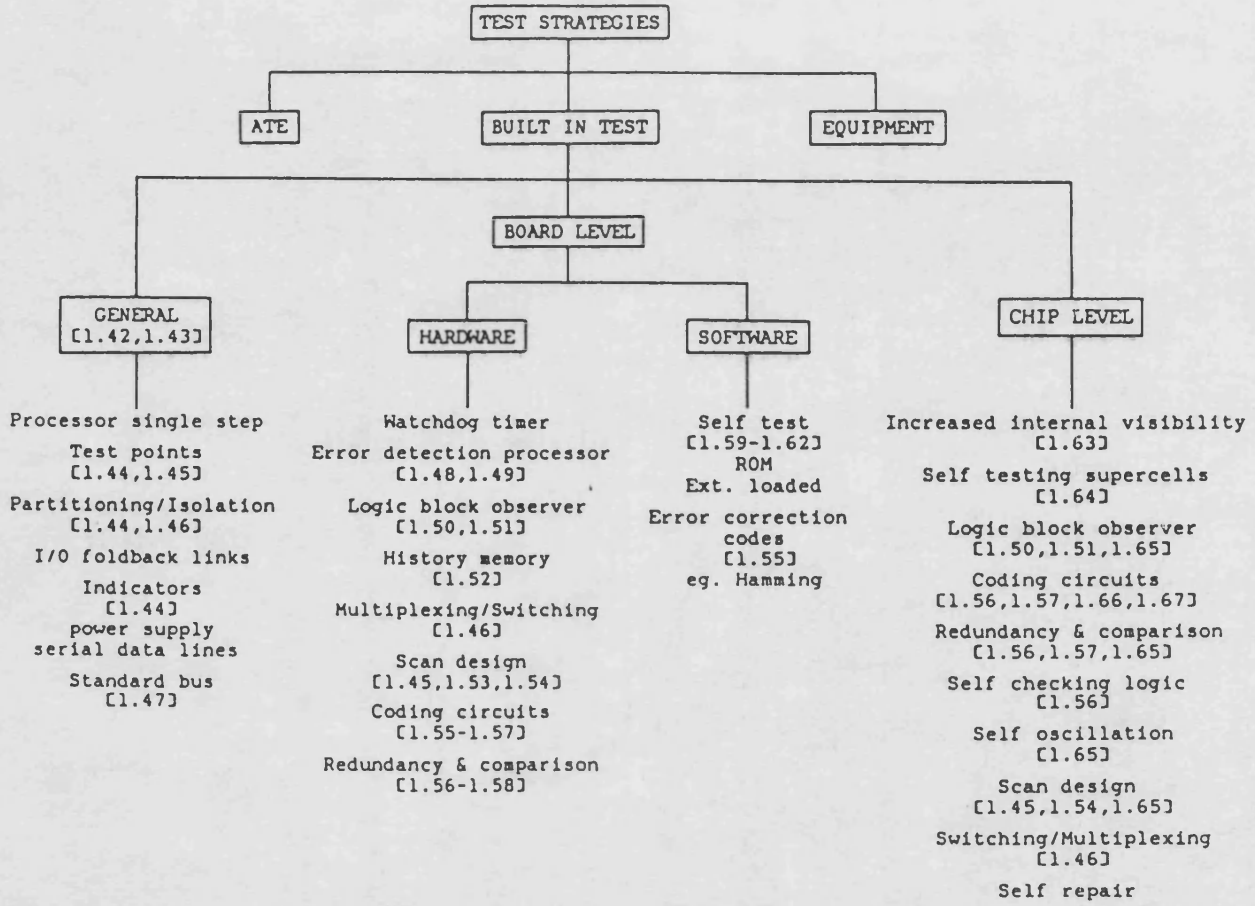


FIGURE 1.1(c) TEST STRATEGIES

CHAPTER TWO : EXISTING TECHNIQUES AND EQUIPMENT

2.1 : SIGNATURE ANALYSIS

Introduced in 1977, signature analysis (SA) compresses a data stream into a four digit signature by means of a sixteen bit shift register with linear feedback [2.1], as shown in Fig. 2.1. This signature represents the nodal activity during the defined period of measurement and is virtually unique. A defined and repeatable stimulus must be created at each node, this being derived from one of three sources, as indicated in Fig. 1.1.

Having located a faulty board, the signatures of the various nodes on that board are observed and compared with those obtained from a known good board, as shown in Fig. 2.2. This allows a fault to be located to component level. The technique can detect all single bit failures and 99.998% of multi-bit failures [2.2]. However, the signature contains no diagnostic information whatsoever, it is purely a go/no-go test.

Limitations of the technique are as follows:

1) Testing the CPU requires on-board provision to break the data bus and force an instruction which causes the address bus of the processor to cycle; for example, a no-operation.
2) All other feedback loops must have a means of being broken, in order to prevent an unstable signature caused by a fault propagating around that loop.
3) If the stimulus is derived from on-board software, then part of the system ROM will have to be allocated for this purpose and hardware added to provide the start, stop and clock signals required by the analyser [2.1].
4) SA cannot cope with asynchronous events, so operations such as direct memory access or interrupts must be synchronously driven, or tested independently.

5) Normally static nodes must be activated. This generally means more hardware, which could itself be a source of failure.
6) SA cannot handle analogue signals.
7) SA cannot cope with timing or noise problems.
8) Considerable time might have to be spent in documenting signatures from a known good board, deriving troubleshooting trees and creating fault dictionaries.

SA can thus diagnose faults to component level, but it may require additional software and hardware to accommodate it. However, it is usually necessary to locate the fault to a board or a module first, so that signatures do not have to be taken from every node in the system. The technique can be considered as the digital equivalent of analogue signal tracing, whereby signatures, not waveforms, are compared with those on the circuit diagram, see Fig. 2.2.

One of the main attributes of SA is that it can be used by a technician who is not familiar with the system being tested, or, for that matter, familiar with computer fault finding techniques. The technician just needs to know what to run and where to probe, taking appropriate action when an incorrect signature is detected.

The signature analyser itself, in comparison with other equipment, is not expensive, but if a basic analyser is to be used, then the hardware and software required in a system to accommodate it will create additional cost overheads. This hardware and software has to be considered at the design stage of a system and prevents SA being applied to systems already in use (circuits cannot easily be added). However, this problem is overcome by using in-circuit emulation as the circuit stimulation. An instrument is available which combines the two techniques [2.3].

Recently, Hewlett Packard has introduced a dedicated instrument to use in conjunction with a standard analyser,

which provides the stimulus, a set of pre-programmed tests (one of these is a processor test) and provision for dedicated tests [2.4]. However, it is not disclosed how they effect circuit partitioning and the breaking of feedback loops, without modification to the board being tested.

Literature tends to indicate a widespread use of SA [2.5], but of the twelve companies surveyed, only one was using the technique. So far, however, this company is not using it for production testing. It is still being evaluated in development circles. Results so far according to them are good. Good devices have been replaced with those having faults and the faulty devices successfully located using SA. Their design will take into account SA, by, for example, allowing a 'watchdog' timer to be disabled during SA evaluation, a circuit which will have to be checked with conventional techniques.

The reaction from another company was that they would never use the technique, because 'the stimulus has no correlation with system operation, it is just a random stimulation'.

Generally though, the technique should be viable for products manufactured in hundreds off, which can more easily justify the time to document signatures and consideration during their design. However, as indicated later, companies producing large quantities still tend to be using automatic test equipment.

2.2 : LOGIC ANALYSIS

A logic analyser (LA) is probably the most powerful tool for all aspects of digital logic, particularly development, debugging and fault diagnosis. However, it is not often used for production testing at present.

Logic analysers essentially fall into two categories:

1) Logic State : Used principally for the observation and debugging of software, but also used to locate hardware failures which cause software errors. Fig. 2.3 shows how information is displayed on a typical logic state analyser.

2) Logic Timing : An extension of a conventional oscilloscope which can generally display up to sixteen waveforms of nodal activity on its screen, but where any point on a waveform can only occupy one of two levels, a logic '1' or a logic '0', as shown in Fig. 2.4. This allows the relative action of waveforms to be observed and precise timing measurements to be made. It is therefore a tool for hardware evaluation, debugging and testing.

It is now usual to find both types of machine combined into one instrument. Certain logic analysers are currently becoming even more powerful, due to the inclusion of a digital oscilloscope within their frame [2.6].

The power of the instrument is a result of major developments during recent years and lies with the features it can offer, which include:

1) General:
   a) Microprocessor controlled.
   b) Menu driven from a keyboard.
   c) Clocking rates up to 500MHz.
   d) Synchronous and asynchronous clocking simultaneously.
   e) Can handle multiphase clocks and multiplexed buses.
   f) Voltage thresholds for TTL, MOS and ECL, or variable.
   g) Comparison/reference memory for state and timing.
   h) Operating parameters and memory can be stored externally.

i) Can handle standard buses, eg. IEEE488 and RS232.

j) Has self and probe tests.

2) State Analyser (synchronous clocking):

a) External clock and control inputs so that the LA samples at the same time as the processor.

b) Minimum of 32 input channels.

c) Minimum of 64 sampled words.

d) Data formatted in binary, hex., octal, decimal or ascii.

e) Disassembly for a specific processor.

f) Search for a specific word.

3) Timing Analyser (asynchronous clocking):

a) Minimum of 8 input channels.

b) Minimum of 1000 sample points.

c) Traces displayed in pages (for more than 16).

d) Trace labels using ascii characters.

e) Trace order definable, with traces available on more than one page.

4) Signal Conditioning:

a) Sample - samples at a clock edge.

b) Glitch - detects threshold transitions between its sample points.

c) Latch - allows the definition of complex timing arrangements between the clock and control signals.

d) Demultiplex - given appropriate control signals, multiplexed buses are automatically demultiplexed.

5) Trigger Modes:

a) Input pattern recognition.

b) Post and pre-trigger.

c) Trigger qualifiers.

d) Nested or sequential trigger levels.

e) Arm and trigger.

f) Delay by events or count.
g) Pass counts.

Despite all the features listed above, the use of logic analysis still seems to be reserved for design, development, debugging and special test benches.

Of the twelve companies surveyed, most had some form of LA but only used it in the testing process for systems which defied fault analysis by any other method. Logic analysis for example, is the only method to resolve problems such as noise, timing and general device incompatibility.

The reluctance to use logic analysers for production testing is attributed to:

1) The complexity of the instrument.
2) The infinite number of possible connections to the system under test.
3) Difficulty of incorporation into an automated test procedure.

In order to use its full capacity, a good working knowledge of the machine and its facilities is essential. If an error in a system is to be rapidly eliminated then the area of the fault must be known, along with the best connection of the analyser probes to locate it. A good knowledge of the system being tested is therefore also required. The operation of a LA is fairly straightforward to master, especially if all information is entered via a keyboard in sequenced menus which indicate when an input error has been made. In addition, help or operating manual pages within the machine greatly assist its use.

The level of operator interaction required with a logic analyser, which has already been reduced, will continue to decrease with:

1) Automated loading of set up parameters and reference memory from a host computer, or from an integral floppy disk unit.
2) Guided probe(s) analysis.

2.3 : AUTOMATIC TEST EQUIPMENT

Automatic test equipment (ATE) is the most common form of board and system testing. Two principal techniques are used with ATE; in-circuit testing and functional testing.

2.3.1 : In-Circuit Testing

In-circuit testing consists of a 'bed of nails', a set of spring loaded contacts pressed onto the wiring side of a printed circuit board by a vacuum, with (ideally) one nail for every circuit node on the board. There is then electrical access to every node in the circuit.

Bare boards can be tested for open circuits and short circuits before assembly. Simple components such as resistors, capacitors, diodes, transistors and op-amps can be isolated with the use of guarding techniques [2.7] and then easily checked. Circuit models are required for more complex integrated circuits. These models would be truth tables for the basic logic gates. Howevever, for certain VLSI devices, such as microprocessors, circuit models are not easily obtained, even from the manufacturer.

Unless standardised (PCB designed on a matrix), a unique bed of nails is required for each board tested. These are expensive and, therefore, hard to justify for low volume production. However, in spite of all cost considerations, it is the ease of fault isolation, due to the internal visibility and controllability of the board via its bed of nails, which makes the technique so attractive.

A device is tested by forcing its inputs to a known state from the application of fast high current pulses which

override existing states, whilst simultaneously monitoring its output(s), as shown in Fig. 2.5. Lines can also be overdriven to isolate or partition components. The rate at which test data can be applied is limited by the interface electronics used for the bed of nails and the quantity of data which has to be processed.

Thus an in-circuit test checks individual component performance and board workmanship but cannot detect dynamic or interactive characteristics. It should also be able to handle both mixed logic and analogue electronics.

2.3.2 : Functional Testing

In a functional test, the board is exercised via its edge connector(s). Fig. 2.6 shows the structure of a typical tester. Stimuli for the board can be generated in a number of ways:

1) Software simulation : The components and connections on the board are described to the software, which will produce a gate level equivalent model (although the availability problem of certain device models arises again). The simulator then generates test programs, patterns and reference data.

2) Vectors : Fixed repeatable pseudorandom patterns are used to generate input vectors. The resulting output vectors are then compared with those from a fault free board.

3) Synthesis of address and data patterns : A programmable tester is used for this purpose.

As the complexity of a board increases, it is difficult to adequately test it solely from its edge connector, so functional testing is usually assisted by guided probing. This part of the technique uses a hand held probe to increase the depth of penetration and observation on the

board. Probe sequences are generated by circuit simulation.

The fixture cost for functional testing is low and a board with good visibility allows faults to be isolated accurately and quickly. Digital and analogue boards can be tested.

2.3.3 : In-Circuit and Functional Testing Combined

Combining the two techniques, as illustrated in Fig. 2.7, overcomes the shortcomings of each. Stimulation is provided as above, whilst the circuit response is checked by signature analysis, a stored correct response, in-circuit testing or probing.

The level of fault detection achieved by combining the techniques is better than either used independently. In addition, an effective test for timing has been achieved in one system [2.8].

Segmentation of the circuit and component isolation is achieved via the bed of nails on which the automatic probing is carried out. The results can then be applied to a fault isolation tree and the suspected failure indicated on a VDU/terminal.

A typical strategy for testing a microprocessor based system is:

1) Standard short circuit test.
2) Test discrete components.
3) In-circuit test of microprocessor.
4) Bus check for shorts.
5) Sequence address bus to confirm address decoding.
6) Functionally exercise CPU with another device:
   a) RAM.
   b) ROM.
   c) I/O.
   d) VDU controller.

General drawbacks of ATE are:

1) Low throughput.
2) Poor test quality.
3) High cost.

2.3.4 : ATE In Use

Major computer manufacturers use ATE almost exclusively to test their boards at a central base. Boards having faults which are unresolvable using ATE are scrapped, simply because it is not cost effective to test further.

The only company surveyed which had volume production uses ATE extensively throughout their manufacturing operations. Devices such as microprocessors and associated programable chips, are not evaluated using ATE, but after the rest of the board is tested, these are inserted and normal application software run to verify them.

One company, whose boards are well defined for documenting and testing, is in the process of ATE installation but another said that the volume of their product could not justify sophisticated ATE, since 'it would not be cost effective'. Yet another was using in-circuit testing via edge connectors, EPROM and processor sockets, followed by a system test.

In other quarters, extensive use is made of analogue and digital models, with the philosophy that if all the interconnections and components are faultless, then the board must work - the design says so. PCBs there are designed around a 0.1" matrix, using computer aided design to allocate components, test points, isolation paths and so on. The computer also produces the PCB layout and jig design details. Their strategy is:

1) Interconnection test.

2) Component checks using static models.
3) Components checks using dynamic models.
4) Components checked in clusters.
5) Functional test of system as in normal operation (no testing as a general computer).

2.4 : SELF TEST

Probably the most important form of testing is self test; software written to interrogate the various modules of a system and to report on failures found. Alternatively, it can be used as a go/no-go test with zero diagnostic information. It is used in just about every stage of every product, from design and development, through to production testing and field testing.

The software can be written to test the system as a generalised computer, or written to test only those operations used in the application program. A typical strategy for the former is given in Fig. 2.8. It need not, however, be restricted to testing but can also be used for performance verification and auto-calibration.

Self test programs can be run:

1) On power up to provide an automatic confidence check.
2) On line: a) as a foreground task.
   b) as a background task.
3) Off line.

The tests can provide rapid go/no-go diagnostic checks for each module in a system, or exhaustively verify the complete system; for example during test by the manufacturer where comprehensiveness is more important than testing time.

The software itself can be:

1) Resident in ROM.
2) Down loaded from a host computer using:
   a) DMA.
   b) RS232.
3) Loaded from disk.
4) Loaded from cassette or paper tape.

Advantages of self test include:

1) Tests are easily generated.
2) Modules can be quickly tested, one after another.
3) No test probes to move around.

The fundamental limitation of the technique, however, is that a certain amount of the system, CPU, ROM (possibly RAM as well) and associated buses, must be fully functional before the program will run at all. Once this has been established, then self test is a powerful tool. It means that devices can be readily exercised in conjunction with others at their normal operating speeds. Given a faulty component or module, then specific programs can be written to probe further, obtaining more information about the fault.

The fundamental limitation of self test, detailed above, can be overcome by running the programs from an external source via buffers for isolation. This falls into the domain of in-circuit emulation, which is considered next.

If the software is stored in on-board ROM, then this creates an additional system overhead but a minimum of external connections are required.

Another area of concern with self test programs is their fault coverage; i.e. the type and quantity of faults they are able to detect. Ideally, all faults should be detected. This problem is analysed by Tasar [2.9], who concludes that the level of fault coverage is dependent on how well written the programs are to start with.

The major computer manufacturers have test software resident in ROM, on dedicated diagnostic boards, loaded from disk, or transmitted along a telephone line from a remote diagnostics centre. Some of this software is extremely powerful, locating faults to component level.

One company has a system consisting of a master controller connected to subcontrollers via an RS232 link, where each controller has its own microprocessor. Test software is resident in each module, but additional software can be downloaded from the master to the subunits via the RS232 link when it is required.

Another form of maintenance strategy used by several companies is the line replaceable unit (LRU). This is a unit which runs tests to check itself and other modules in background and/or foreground modes, indicating any fault condition(s) found on lights or a dedicated message display. When a fault occurs the LRU can be easily replaced. All the individual boards of the faulty LRU will be tested, sometimes on site but more generally at a central repair base, proceeding to a system test if the fault is not found and finally being given to the boffins to sort out in sheer desperation.

Standard checks of CPU, ROM, RAM and I/O may be carried out in a background mode, whilst checks on current levels and other vital parameters are carried out in the foreground. Some companies prefer to perform a series of 'actual' tests, checking the system as it actually operates and not as a general computer.

2.5 : IN-CIRCUIT EMULATION

Microprocessor emulation is the most common form of in-circuit emulation (ICE) in microprocessor based systems, since the processor is the focus for all system functions. The operation of the system processor is emulated by a

similar processor in the ICE equipment, which is connected via a cable to the socket of the system processor. Fig. 2.9 shows a typical in-circuit emulator and its connection to the system under test (SUT). No modifications are required to the SUT. The processor in the emulator is totally controllable which, together with the resources of the instrument, allows hardware and software to be easily evaluated and debugged. It is therefore a powerful development tool.

Operationally, ICE is similar to ATE in that test patterns or vectors are generated and the response of the circuit to them is observed.

Generally only the clock of the SUT must be functional to carry out ICE, so diagnostic software in the emulator can exercise all components in the SUT, isolating faults to module or subsystem level.

Typical features of ICE are:

1) Real time transparent emulation of processor.
2) Control over CPU registers, I/O ports and memory.
3) Breakpoints.
4) Menu driven.
5) Subsequent run comparison (reference memory).
6) Trace memory.
7) Write protection.
8) Frequency and pulse measurement.

Emulators typically have three operating modes:

1) Interrogation:
   a) Display and modify CPU registers, including program counter and stack pointer.
   b) Display and modify memory and I/O ports.
   c) Set breakpoints for a number of different conditions.
   d) Display trace details from high speed memory.

e) Display measured execution time.

2) Run/Emulation:
   a) Execute test programs, testing and comparing information.
   b) Display register or memory contents, along with address or data bus states on a sampled basis.

3) Single Step:
   as for 2), but one cycle at a time.

Emulation must be able to cope with all current and future processors and in particular be able to handle:

a) Interrupts.
b) DMA.
c) Memory refresh.
d) Multiplexed buses.
e) Illegal accesses.
f) Asynchronous communication.
g) Multiprocessors.
h) Cache and queue memory.
i) Instruction prefetch.

In addition, with systems supporting in excess of one mega-byte of memory and, therefore, making extensive use of memory managers, system emulation must now be considered, whereby the processor and the memory manager are emulated together to distinguish between logical and physical addresses. System emulation must also cope with shared memory [2.13].

Advantages of ICE are:

1) Software is written in the assembly language of the target processor (or even a higher level language).
2) The system is tested as used (CPU instructions), instead of a meaningless stream of bits.
3) A working kernel is not required (the minimum amount

of CPU and ROM necessary to run a program).

4) A system overhead is not created for testing (no additional system ROM).
5) Minimal test fixtures are required (no bed of nails or edge connector).
6) Hardware and software can be tested and debugged under tightly controlled conditions (eg. start,step,stop).
7) Tests can be executed in real time at normal or abnormal clock rates.

Limitations of ICE are:

1) A socketed CPU is required, which could be a source of unreliability. This is especially a problem in military applications.
2) Problems are generally located to module not component level due to a lack of visibility.

As far as the use of ICE in industry is concerned, the company survey revealed that, like logic analysis, it seems to be reserved for development use only.

Extensive ICE has been practically implemented on a specific microprocessor based system, using a Millenium Microsystem Analyser (in-circuit emulator). This work is detailed in Appendix A.

2.6 : COMBINED TECHNIQUES

All the techniques considered so far have limitations as indicated. The tendency now is to combine techniques in one instrument, so overcoming at least some of the inherent problems of each. The combination of logic state and timing analysis in one instrument is already standard, with the combination of in-circuit and functional ATE becoming so [2.10]. Other techniques available together are:

1) ATE with self test [2.11].

2) ICE with SA [2.3,2.12].
3) ICE with logic state analysis [2.13].
4) ICE with complete logic analysis [2.14].
5) Logic analysis with pattern generation (ATE) [2.15, 2.16].
6) Logic state analysis with SA [2.17].

In a previous paper [2.18], it was suggested that ICE and logic analysis would be the most effective combination. Individually, these two techniques are not widely used for production testing but they are nevertheless powerful diagnostic tools. This is still believed to be the case for the techniques considered so far, with proposed operation as follows. Tests and stimuli generated by the emulator would be monitored by the LA using guided probes. State and timing information could then be recorded automatically and compared with a set of expected responses. Resulting error information should be displayed in state, timing and English formats. The LA would also be available to deal with time related failures and glitches. Instruments are currently available which combine these two techniques, but they do not function exactly as outlined above.

Self test, as has already been indicated, appears at some point in the life of every processor based system, whichever overall single or combination of techniques is used to test it. It will also continue to do so, since in terms of generation, implementation and cost it creates minimal overheads.

Test instruments developed to cope with increased circuit complexity are certainly becoming more sophisticated. One such instrument [2.19] runs a 'learn' algorithm to interrogate the system and generates a memory map of all devices found. It then proceeds, based on this information, to run a series of pre-programmed tests. Also part of this equipment is a hand held probe, which can observe nodal activity or provide stimulus in synchronism with

operations in the system under test.

2.7 : PROVISION FOR TEST

Design for testability has become a major requirement for all microprocesor based systems, especially those diagnostic techniques which require built in provision to use them. This built in provision for test may vary from special test points or memory for test software, to removable wire links or additional gates to allow external control of devices. In this chapter it has been built in test (BIT) at board level for subsequent external testing. More significantly now and certainly in the future, is BIT at board level for internal (self) testing and BIT at chip level for self checking chips. These techniques (see Fig. 1.1) are considered in the next Chapter.

2.8 : REFERENCES

2.1) Fault diagnostics for microprocessor based systems - T. J. Hollis; Project report for the degree of B.Sc. in Electrical Engineering; University of Bath, England; 1980; pp. 13-15.

2.2) Instruments - A. Santoni; EDN; July 20,1979; p. 97.

2.3) Combining diagnosis and emulation yields fast fault finding - L. Badagiliacca, R. Catterton; Electronics; November 10, 1977; pp. 107-110.

2.4) Applying signature analysis to existing processor-based products - R. Rhodes Burke; Electronics; February 24, 1981; pp. 127-133.

2.5) Signature analysis wins new acclaim - M. Marshall; Electronics; February 14, 1980; pp.103-4.

2.6) Logic analyzer model 1631/AD...the one with the scope; Data sheet 5954-2614; Hewlett Packard; USA; 1985.

2.7) 3062A board test system; Data sheet 5953-6934; Hewlett Packard; USA; 1983.

2.8) Test LSI boards functionally on an in-circuit tester - T. Jackson, P. Vais; Electronic Design; October 29, 1981; pp. 137-144.

2.9) Analysis of fault detection coverage of a self-test software program - V. Tasar; FTCS-8*; pp. 65-71.

2.10) ATE swings toward merged in-circuit, functional tests - J. McLeod; Electronic Design; October 29, 1981; pp.115-9.

2.11) Quality and control self-test - R. W. Spearman, F. D. Patch; 1980 Test Conf.*; pp.279-286.

2.12) Plan a mixed strategy to simplify uP systems tests - B. Hordos; EDN; April 20, 1979; pp. 183-187.

2.13) Emulator solves system-level debug problems - R. Parks, K. Greenberg; Electronic Design; May 14, 1981; pp.173-180.

2.14) Analysers, emulators: Together they stand for better testing - R. Allan; Electronic Design; June 25, 1981; pp 75-92.

2.15) Flexible pattern generation aids logic analysis - S. Palmquist, M. Pettet; Electronic Design; October 15, 1981; pp. 165-174.

2.16) Logic analyser delivers test patterns too - S. Palmquist, G. Hoeren; Electronics; September 8,

1981; pp.113-9.

2.17) Logic state and signature analysis combine for easy testing - I. H. Spector; Electronics; Vol. 51, No. 12; June 8, 1978; pp. 141-5.

2.18) The maintainability of processor systems - T. J. Hollis; Microtest '81; 21-24 September 1981; University of Bath, England; Society of Electronic and Radio Technicians; pp. 110-123.

2.19) Portable tester learns boards to simplify service - D. Cassas; Electronics; June 16, 1981; pp. 153-160.

* see section B.5.

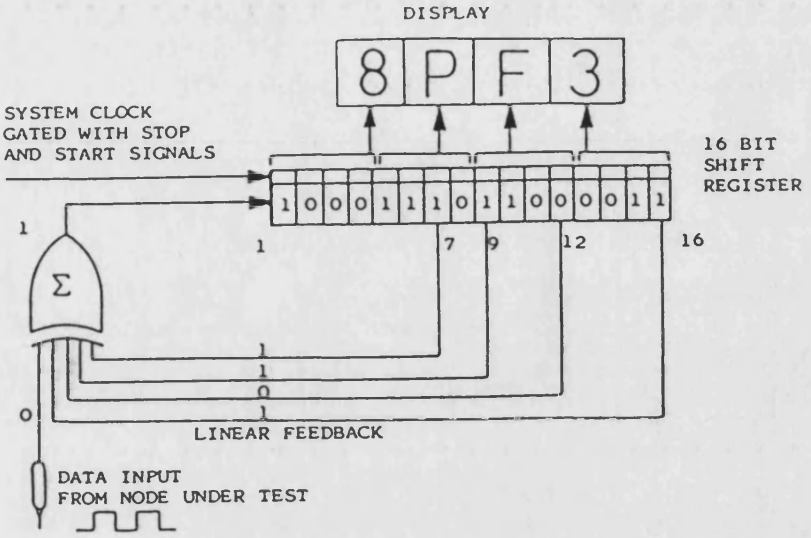


FIGURE 2.1 THE PRINCIPAL CIRCUIT OF A SIGNATURE ANALYSER

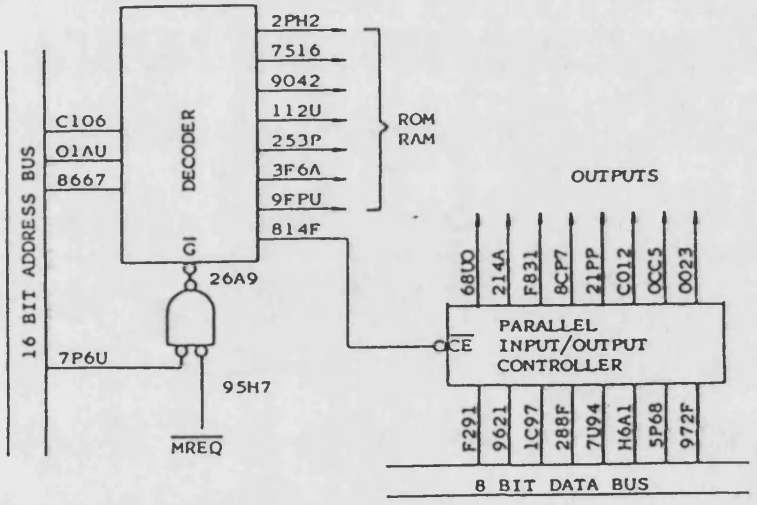


FIGURE 2.2 CIRCUIT DIAGRAM WITH SIGNATURES

TRIG	0D00	10100	3A
	0D01	10101	10
	0D02	10101	09
	0910	10101	47
	0D03	10100	3C
	0D04	10100	32
	0D05	10101	10
	0D06	10101	09
	0910	10011	48
	0D07	10100	18
	0D08	10101	F7
	0D00	10100	3A
	0D01	10101	10
	0D02	10101	09

ADDRESS STATUS/ CONTROL DATA

FIGURE 2.3 TYPICAL DISPLAY ON A LOGIC STATE ANALYSER

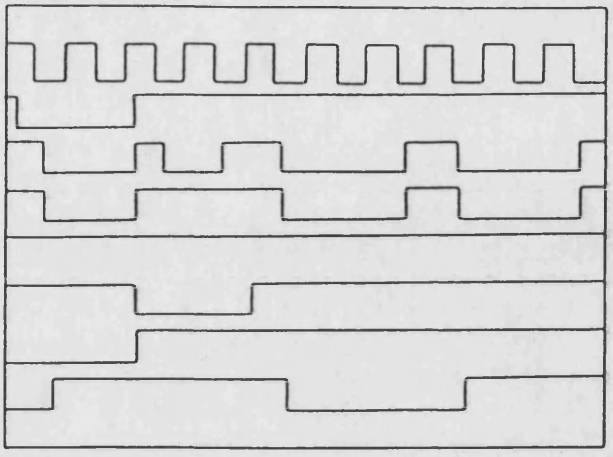


FIGURE 2.4 TYPICAL DISPLAY ON A LOGIC TIMING ANALYSER

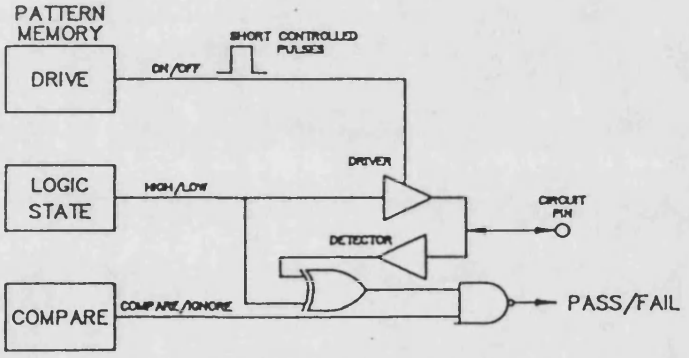


FIGURE 2.5 OVER-DRIVING A CIRCUIT PIN OR NODE

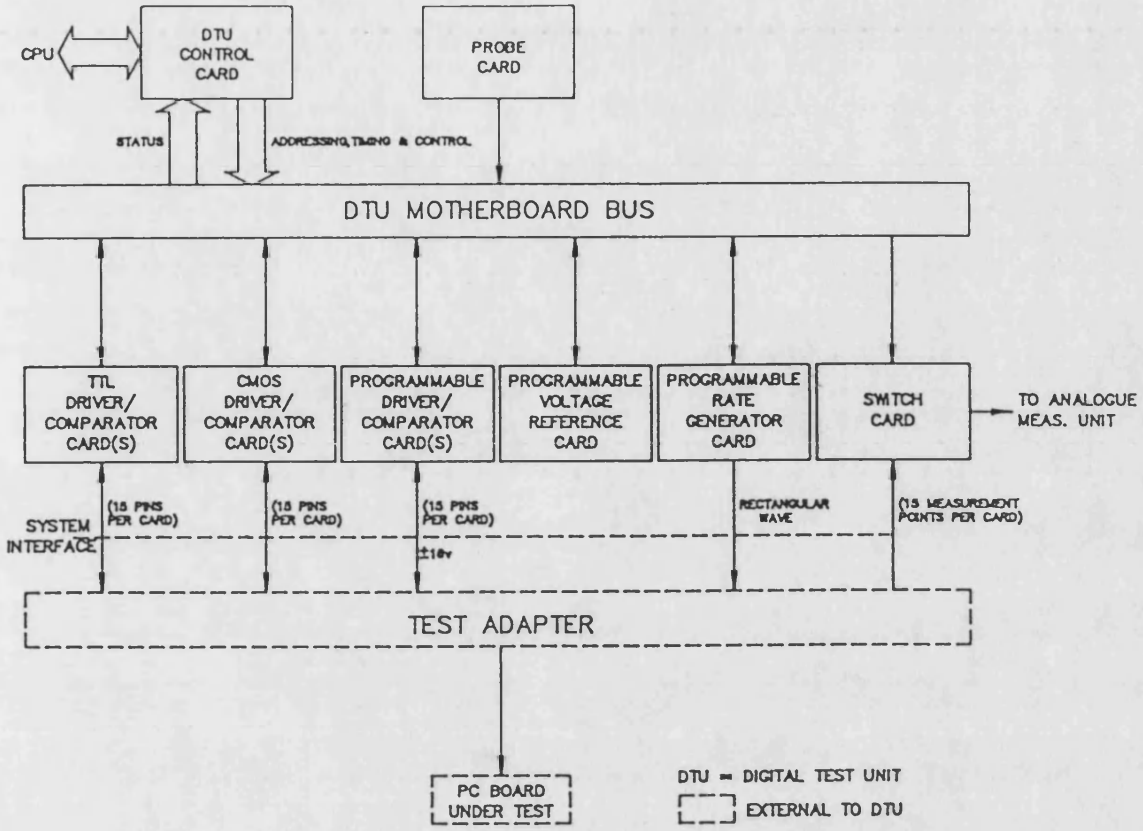


FIGURE 2.6 STRUCTURE OF A TYPICAL FUNCTIONAL BOARD TESTER.

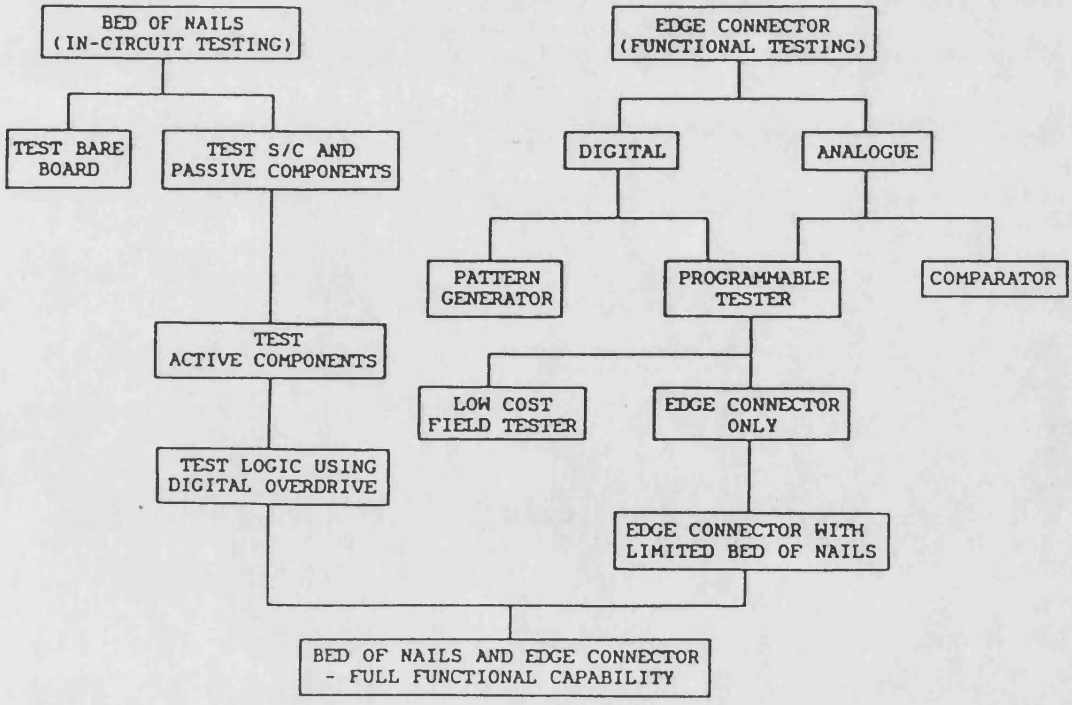


FIGURE 2.7 THE COMBINATION OF IN-CIRCUIT AND FUNCTIONAL TESTING

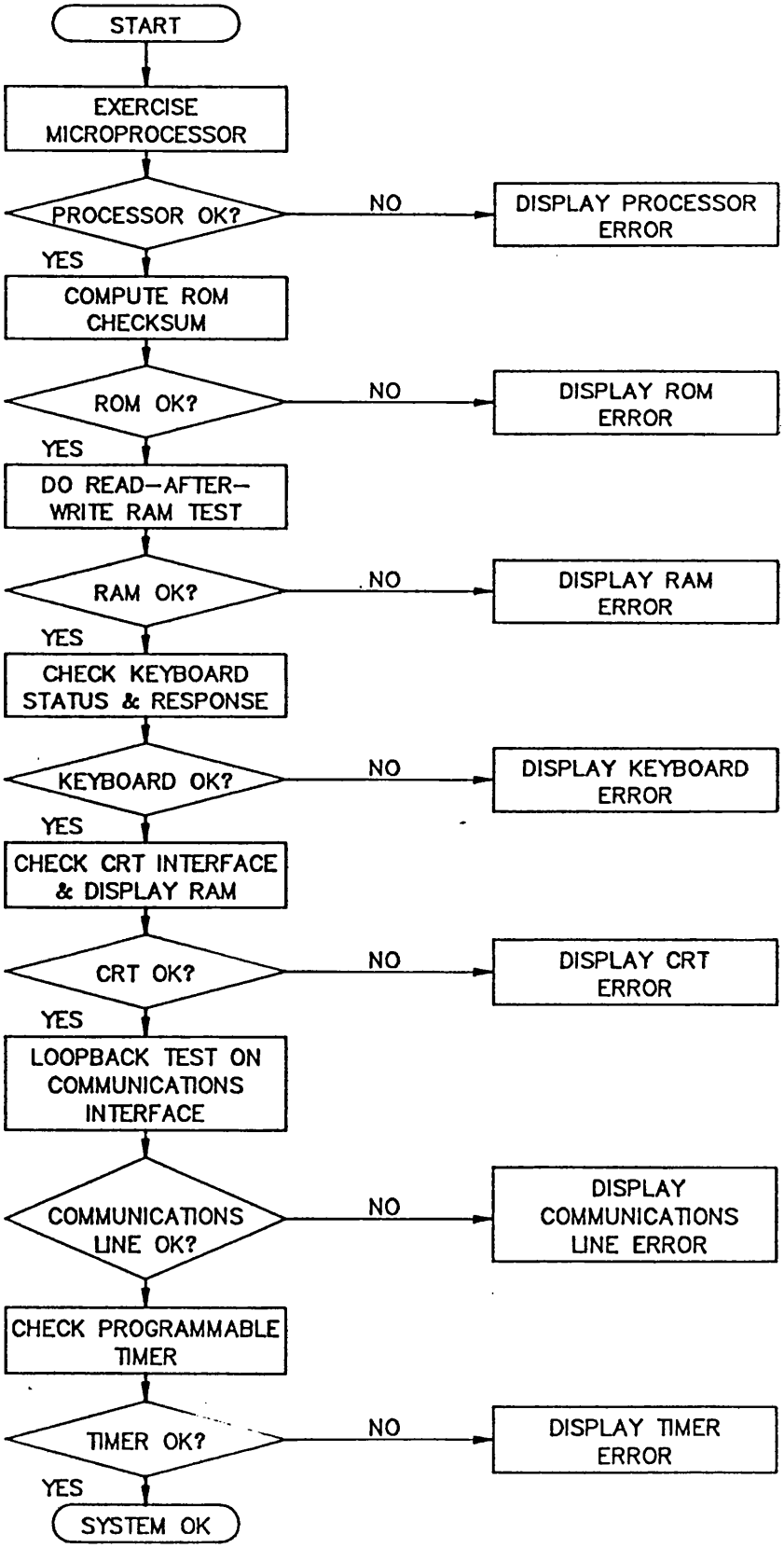


FIGURE 2.8 A COMPLETE SELF TEST PROGRAM

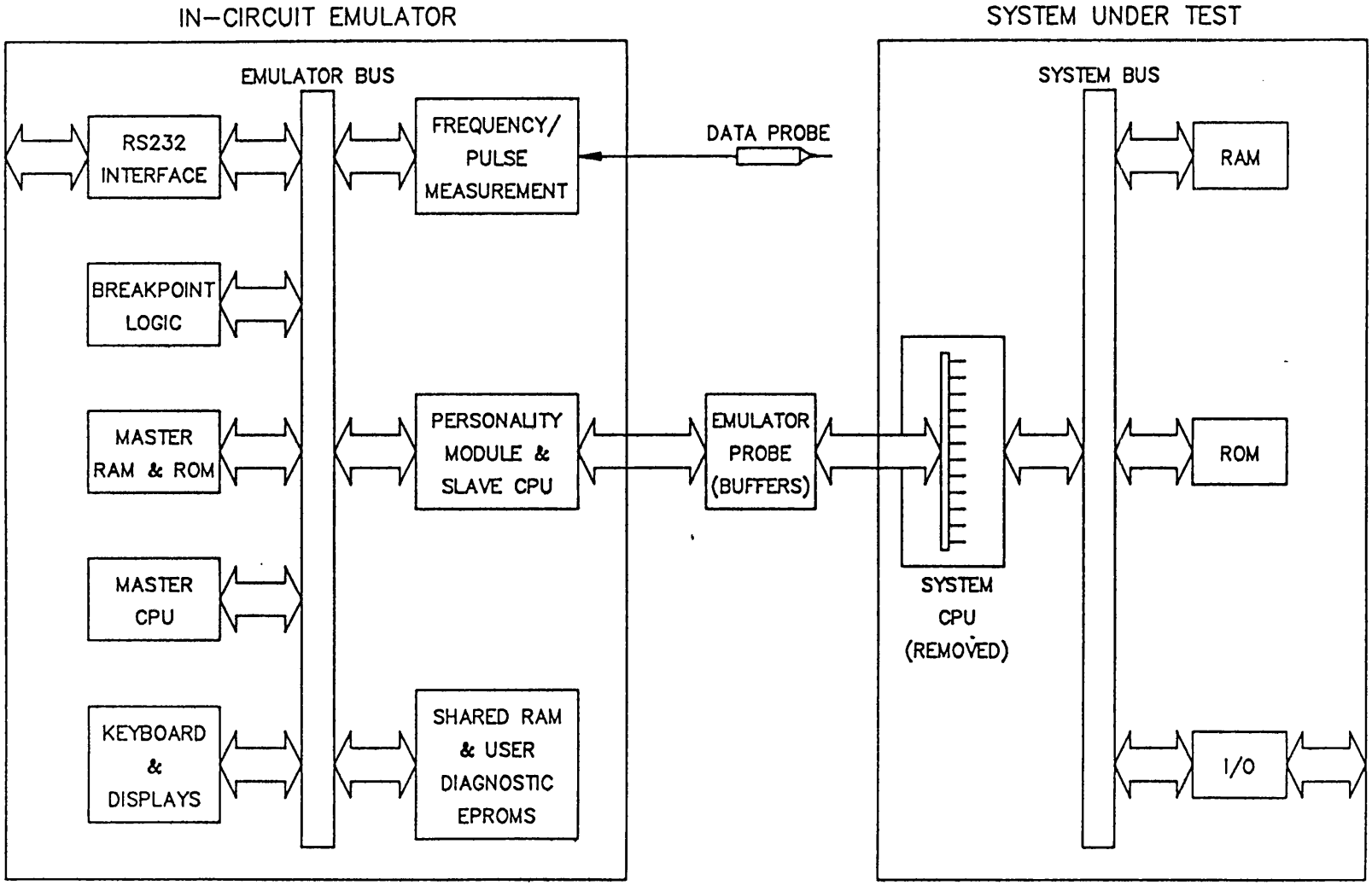


FIGURE 2.9 STRUCTURE OF A TYPICAL IN-CIRCUIT EMULATOR AND ITS CONNECTIONS

CHAPTER THREE : BUILT IN TEST - AN INTRODUCTION

Microprocessors are typical of VLSI circuits, so a lot of attention has been given to their testing [3.1-3.15]. Devices in a microprocessor based system are difficult to test because:

1) The number of possible faults is extremely large. There are thousands of gates and interconnecting lines, all subject to failure, and thus a successful test will require a large number of test patterns.
2) Access to internal components and lines is severely limited by the number of I/O connections; say 5000 gates but only 40 I/O pins.
3) New and complex failures such as pattern sensitivity occur.
4) A complete description of the device is generally not available. Microprocessor specifications, for example, typically consist of a register level block diagram, a listing of the instruction set of the processor and some information on system timing.

These points highlight the problem areas that built in test (BIT) must overcome. This chapter introduces four concepts to describe and specify BIT techniques. These are:

1) Faults and Fault Models - the types of fault to be detected, and their effect on circuit operation.
2) Observability and Controllability - the level of access to internal components.
3) Design for Testability - the mechanisms to implement during design.
4) Redundancy - the quantity of additional hardware and software for test purposes.

3.1 : FAULTS AND FAULT MODELS

Faults fall into two categories; permanent (solid) and

intermittent (transient). Intermittent failures are the most difficult to test for and can be caused by timing and noise problems, or clock waveform and power supply degradation. A test sequence which will detect solid faults in one pass may have to be repeated many times to stand any chance of detecting intermittent failures.

A combinational circuit may be tested by applying all possible input combinations (test patterns) to verify its truth table. If the circuit has twenty inputs, for example, then this means 2^{20} = approx. 10^6 patterns. However, all single and multiple faults will be detected (assuming they are detectable).

Sequential circuts require a minimum of $s2^n$ test patterns, where s is the number of stable states, which must be arranged as a specific checking sequence, a difficult and time consuming task.

The quantity of test patterns and therefore test time for circuits with a significant number of inputs is large. To reduce both, it is necessary to test for a subset of all possible faults. A list of likely failures is constructed, fault models created to represent the logical characteristics of these faults, input patterns generated to test for them (test pattern generation) [3.16-3.18] and test verification implemented using fault simulation [3.16] to determine the faults actually detected (fault coverage).

Faults and fault models as applied to VLSI are now considered.

3.1.1 : Stuck-At Fault Model

The stuck-at fault model assumes that a logic gate has an input or output permanently stuck at either a logic 0 or a logic 1. Fig. 3.1a shows a fault free AND gate, whilst Fig. 3.1b shows the same gate with a stuck-at-1 (SA1)

fault. The faulty gate sees input A as a 1 irrespective of the actual level applied to it.

A circuit might not physically have a stuck-at failure (a node literally connected to 0V or the positive power rail), but exhibits the characteristics from an operational point of view. Consider the circuit in Fig. 3.2, a typical NAND gate. If the base-emitter junction of input A ($b\text{-}e_1$) is open circuit, then input A appears to be SA1, whereas a collector-emitter short on TR4 looks like output Y stuck-at-0 (SA0).

If a network has a total of N gate inputs and outputs, then since each of these may be SA0, SA1, or correct, then the total number of fault combinations is 3^N-1. So for a circuit with thirty gates, each having two inputs, there would be approximately 2×10^{14} faulty states. This is far too many faults to assume, as the time required to generate and implement tests for them would be impractical.

In order to overcome this problem, the model of a single stuck-at fault has been assumed for many years, in which a faulty circuit is considered to have one single stuck-at failure only. Thirty 2-input gates will then have one hundred and eighty possible single stuck-at faults.

The quantity of faults can be further reduced by fault equivalencing [3.16]. Consider the AND gate in Fig. 3.3 with one of its inputs SA0. The effect of this fault is equivalent to the output Y SA0, so for test purposes only one of the two faults needs to be considered.

The single stuck-at fault model might be suitable for SSI and MSI, but LSI and VLSI have other faults (excluding manufacturing defects and design faults [3.19-3.21]) which cannot necessarily be modelled as stuck-at failures. These are detailed next.

3.1.2 : LSI/VLSI Failures

1) Pattern Sensitive Failures : These are failures observed only when a specific set of states (logic levels) or sequence of states occurs. These are particularly prevalent in memory. Example failures are the non recognition of a single 0 (1) following a string of 1's (0's) due to unwanted hysteresis effects, or the generation of noise (crosstalk).

2) Transformation Failures : These are failures which create feedback, so that sequential behaviour or oscillation is exhibited from a combinational circuit. An example of this is the input short illustrated in Fig. 3.4. CMOS is particularly susceptible to these failures.

3) Environmental Failures : These are failures caused by heat, vibration and humidity, as well as electrical and electromagnetic interference.

4) Parametric Failures : These are failures such as resistance or transistor gain variation due to component aging.

A lot of these faults will be intermittent and difficult, if not impossible, to model as single stuck-at faults. In some cases an electrical model might be more appropriate than a logical model. Galiay [3.22] comments on this and details failures related to integrated circuit (IC) construction and layout. He then suggests IC layout rules to assist test, which decrease failure types and avoid those which are difficult to detect.

The inadequacy of the stuck-at fault model for VLSI is considered by Nickel [3.23], who presents nearest neighbour and neighbourhood interaction fault models, principally for memory.

Two other important classes of faults are bridging and unidirectional faults.

3.1.3 : Bridging Faults (shorts)

In general, the stuck-at fault model does not cover bridging faults. If two active lines are shorted together, one at logic 0 and the other at logic 1, the resulting level could be a 0, a 1 or indeterminate. The individual source impedances principally determine which level will win, but in practice either an ORing or an ANDing operation will take place. For the OR situation, a short circuit, as indicated in Fig. 3.5a, can be modelled as an additional gate, as shown in Fig. 3.5b.

Bridging faults in particular are a problem in programmable logic arrays (PLAs) [3.24].

3.1.4 : Unidirectional Faults

Unidirectional multiple errors are those where all erroneous bits are the same; either 1's changed to 0's or 0's changed to 1's. For example, a unidirectional error can change 0110 to 1111 or 0000, but not 1001. The failure of a power lead to or in an IC, such that a number of its data lines become stuck at the same logic value, creates a unidirectional error. Alternatively, a single fault in the address decoding circuitry of a ROM chip could cause the correct word and an incorrect word to be simultaneously selected. Externally, the words would generally appear to be ORed together, resulting in a word with more 1's than the correct word. This fault causes data dependent unidirectional errors, whereas the power lead failure produces a permanent undirectional error. This type of fault can also occur in the transmission of serial data from a device having a single failure.

3.1.5 : Functional Fault Model

Since the information required to generate logical or electrical models is generally not available for VLSI, then a technique proposed by Thatte [3.2,3.11,3.14] makes use of the information which is. For microprocessors, this means using the register level block diagram and instruction information to produce a functional fault model. Rather than testing each instruction with various operands and exercising the internal registers, various functional modules are defined from the architectural organisation and instruction information, which are independent of their actual implementation. Each functional unit is allowed to have one functional fault, such as the wrong instruction or no instruction executed (although these may be caused by stuck-at faults) and test patterns generated to locate these faults.

Marchal [3.25] extends these principles using the single or dual internal buses of a microprocessor.

3.2 : OBSERVABILITY AND CONTROLLABILITY

The principal aim of any BIT technique is to increase the observability and controllability of a network.

If system inputs are fed to an AND gate, as shown in Fig. 3.6a, then these inputs are directly controllable. Similarly, if the output from the AND gate is a system output, then it is directly observable. If this AND gate becomes part of a large combinational circuit, as indicated in Fig. 3.6b, then it is more difficult to control and observe this one gate. Going one step further, if the AND gate is embedded into a much larger sequential circuit (VLSI), then the problem gets much worse.

The testability of a circuit is therefore directly related to the difficulty of controlling and observing internal nodes from system inputs and outputs respectively. Ideally, every node should be controllable and observable.

A method of analysing the controllabilty and observability for combinational and sequential networks has been proposed by Goldstein [3.26].

3.3 : DESIGN FOR TESTABILITY

Built in test (BIT) requires design for testability, a provision for test at the design stage. Design for testability is broadly divided in to two categories; the so called 'ad hoc' approach or the structured approach. This chapter is concerned with the structured approaches, but since these frequently extend the concepts of the ad hoc methods, it is worth listing some of these ad hoc methods first, with particular reference to microprocessor based systems.

1) Partitioning:
   a) Mechanical:
      i) Separate boards for different areas/functions of the system; for example, CPU, memory and I/O cards.
      ii) Removable wire links, either on the board or external to the board. These could be feedback paths which can then be broken.
   b) Logical : Additional logic and control lines to allow direct control of internal modules, see Fig. 3.7, external control of the clock, see Fig. 3.8, or direct observation and control of module interconnections, see Fig. 3.9 [3.16,3.27-3.29]. The latter technique is particularly suited for external control of microprocessor buses, so that perhaps switches can set up the desired information [3.30].

2) Test Points [3.16,3.29,3.31,3.32]:
   a) Additional observation points.
   b) Additional observation/control points (as required for logical partitioning).
   c) Bed of nails (as used for ATE).

3) Indicators:

a) Single LEDs to indicate the state of power rails, serial data lines and general gate/chip inputs or outputs [3.29].

b) Latches and seven segment displays for bus information [3.30,3.33].

4) Microprocessor Single Step : Additional circuitry which allows a microprocessor to execute a single instruction or clock cycle [3.28,3.33].

5) Signature Analysis (see section 2.1).

6) Self Test Programs (see section 2.4).

7) Watch Dog Timer [3.34]: Checks the response of the microprocessor to interrupts, or checks that software or hardware processing time is within a predetermined limit.

8) Control Line Access : Allows normally static control lines to be externally controlled by using tie-up or tie-down resistors [3.31].

9) Standard Bus Structures : Well defined bus structures such as the Microbus [3.35], the G64 Bus [3.36], or the VME Bus [3.37] allow the control and observation of bus activity to be standardised for test purposes.

3.4 : REDUNDANCY

Most of the ad hoc and structured approaches use some form of redundancy to achieve improved testability; i.e. not necessary for normal system operation. Redundancy can be classified as follows:

1) Hardware : Additional hardware for observation (checking) or control.

2) Software : Additional routines such as self test programs.

3) Time : If system checking is performed during normal operation, then hardware can do this with no extra time burden. However, software checking will impose a time overhead (redundancy).

4) Information : Both hardware and software require information redundancy. They need and/or generate more information than is necessary for normal operation.

3.5 : REFERENCES

3.1) Microcomputer for emulation bares hidden buses, functions - J. Moon; Electronics; July 17, 1980; pp. 126-129.

3.2) Test generation for microprocessors - S. M. Thatte, J. A. Abraham; IEEE Trans. Comput.; Vol. C-29, No. 6; June, 1980; pp. 429-441.

3.3) Testability considerations in microprocessor-based design - J. P. Hayes; Computer; March, 1980; pp. 17-25.

3.4) Survey of approaches to testing and diagnosing microprocessor-based systems - W. Schmitt, E. Lynch; Autotestcon '80*; pp. 127-130.

3.5) The microprocesor testing revolution - E. S. Donn; 1979 Test Conf.*; pp. 121-124.

3.6) Design of easily testable bit-sliced systems - T. Sridhar, J. P. Hayes; IEEE Trans. Comput.; Vol. C-30, No. 11; November, 1981; pp. 842-854.

3.7) A testable design of general purpose microprocessors - R. Parthasarathy; FTCS-12*; pp. 117-123.

3.8) Testing the new generation of microprocessors - P. Wohlfarth, D. Smith; 1979 Test Conf.*; pp. 255-257.

3.9) Testing internally clocked LSI - R. Winn; 1979 Test Conf.*; pp. 338-340.

3.10) Bit-slices microprocessors testing - a case study - M. El-Lithy, R. Husson; FTCS-10; pp.126-1.

3.11) User testing of microprocessors - S. M. Thatte, J. A. Abraham; IEEE Compcon '79; Spring 1979; San Francisco, California, USA;IEEE Pub. No. 79 CH1393-8C; pp. 108-114.

3.12) Testing the 8086 - M-G Lin, A. K. Yuen, K. Rose; 1980 Test Conf.*; pp. 426-431.

3.13) A functional test method for microprocessors - L. Shen, S. Y. H. Su; FTCS-14; pp 212-218.

3.14) Fault coverage of test programs for a microprocessor - J. A. Abraham, S. M. Thatte; 1979 Test Conf.*; pp. 18-22.

3.15) Testing using a minimal number of instructions - P. K. Lala; Microprocessors and Microsystems; Vol. 5, No. 7; September 1981; pp. 295-298.

3.16) Design for testability - T. W. Williams, K. P. Parker; Proc. IEEE; Vol. 71, No. 1; January, 1983;

pp. 98-112.

3.17) Test generation techniques - S. B. Akers; Computer; March, 1980; pp. 9-15.

3.18) LSI logic testing - an overview - E. I. Muehldorf, A. D. Savkar; IEEE Trans. Comput.; Vol. C-30, No. 1; January, 1981; pp. 1-16.

3.19) Microprocessor device reliability - E. R. Hnatex; 1977 IEEE Southeastcon Region 3 Conference; pp. 82-86.

3.20) Improvements in reliability of VLSI integrated circuits - M. Noble; IEE Electronics and Power; March, 1985; pp. 222-226.

3.21) Hardware logic design faults - a classification and some measurements - T. L. Faulkner, C. W. Bartlett, M. Small; FTCS-12*; pp. 377-380.

3.22) Physical versus logical fault models MOS LSI circuits; Impact on their testability - J. Galiay, Y. Crouzet, M. Vergniault; IEEE Trans. Comput.; Vol. C-29, No. 6; June, 1980; pp. 527-531.

3.23) VLSI - The inadequacy of the stuck at fault model - V. V. Nickel; 1980 Test Conf.*; pp. 378-381.

3.24) The effect of untestable faults in PLAs and a design for testability - D. K. Pradhan, K. Son, 1980 Test Conf.*; pp. 359-367.

3.25) Updating functional fault models for microcomputer internal buses - P. Marchal; FTCS-15; pp. 58-64.

3.26) Controllabilty/observability analysis of digital circuits - L. W. Goldstein; IEEE Trans. on Circuits and Systems; Vol. CAS-26, No. 9; September 1979; pp. 685-693.

3.27) Design for testability for microprocessor-based boards - R. Willis; Electronic Engineering; April, 1983; pp. 67-72.

3.28) Designing boards for testability - G. Foley; New Electonics; Vol. 12, No. 23; November 27, 1979; pp. 98-102.

3.29) Practical troubleshooting techniques for microprocessor systems - J. W. Coffron; Prentice-Hall Inc, Englewood Cliffs, New Jersey, USA; 1981; Chapter 8.

3.30) Save both time and money debugging a uP prototype board -M. M. Zyla; EDN; 5 September, 1978; pp. 153-156.

3.31) Design for testability - J. Turino; Logical Solu-

tions Inc.; Campbell, Ca., USA; 1979; Chapter 4.

3.32) Enhancing testability of large-scale integrated circuits via test points and additional logic - M. J. Y. Williams, J. B. Angell; IEEE Trans. Comput.; Vol. C-22, No. 1; January, 1973; pp. 46-60.

3.33) Testing microprocessor-based systems at home - C. Carson; Wireless World; April, 1984; pp. 44,45,49.

3.34) Designing fail-safe microprocessor systems - D. R. Ballard; Electronics; 4 January 1979; pp. 139-143.

3.35) Microprocessor bus standard could cure designer's woes - G. Force; Electronics; 20 July, 1978; pp. 113-118.

3.36) The G-64 route to microsystem hardware implementation - P. Wilson, C. Cordingley; New Electronics; 12 November, 1985; pp. 43-46.

3.37) Decentralising uP bus grows easily from 16 to 32 bits - C. Kaplinsky; Electronic Design; 12 November 1981; pp. 173-179.

* see section B.5.

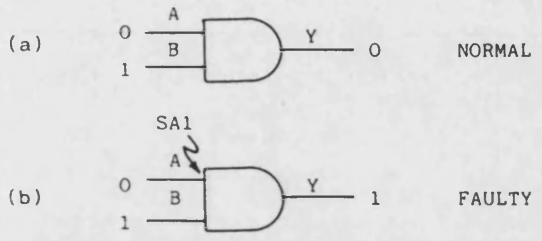


FIGURE 3.1 A STUCK-AT FAILURE

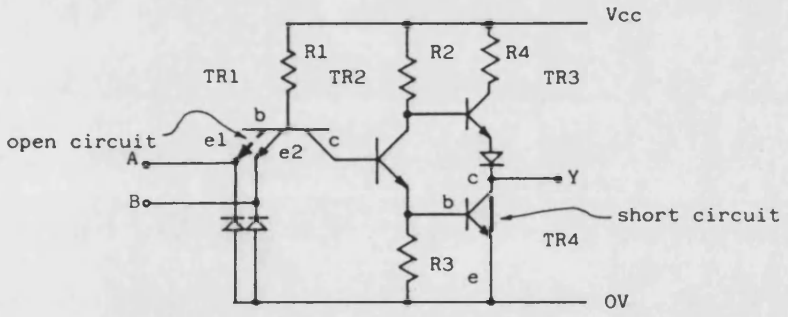


FIGURE 3.2 TWO FAILURES IN A TTL NAND GATE

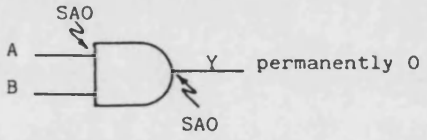


FIGURE 3.3 EQUIVALENT STUCK-AT FAILURES

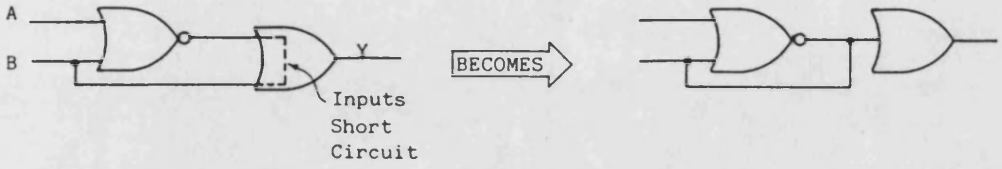


FIGURE 3.4 A TRANSFORMATION FAILURE

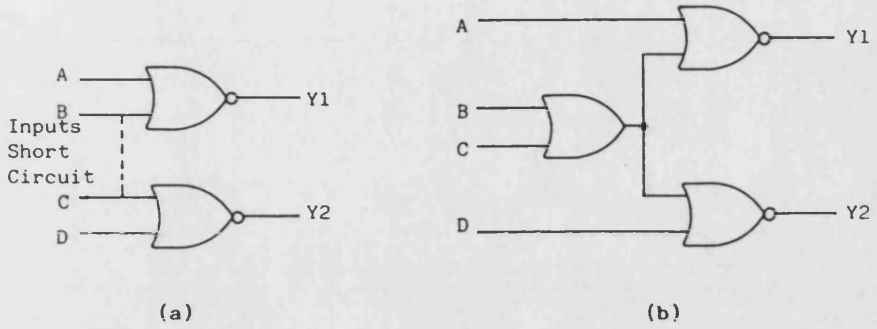


FIGURE 3.5 MODELLING A SHORT CIRCUIT FAULT

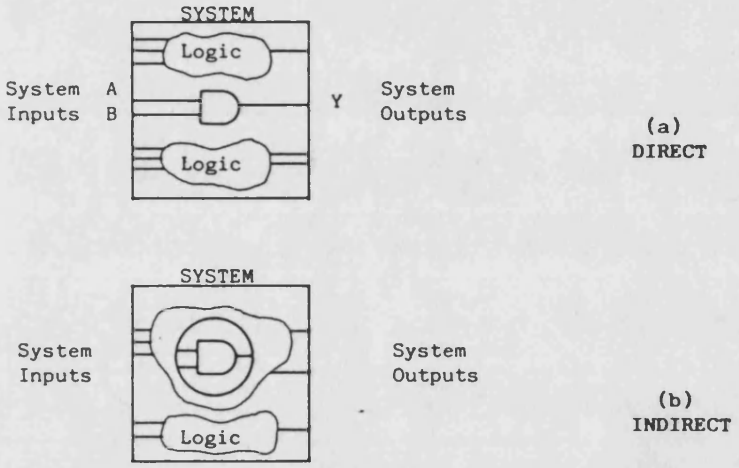


FIGURE 3.6 THE CONTROLLABILITY AND OBSERVABILITY OF A SINGLE GATE

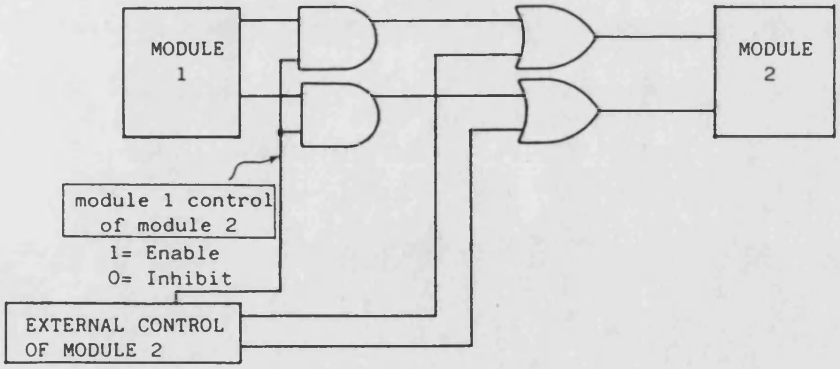


FIGURE 3.7 DIRECT CONTROL OF AN INTERNAL MODULE USING ADDITIONAL LOGIC

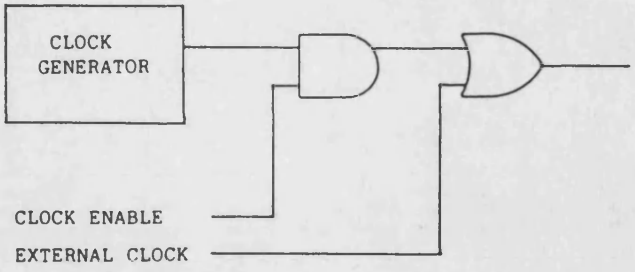


FIGURE 3.8 EXTERNAL CONTROL OF A SYSTEM CLOCK

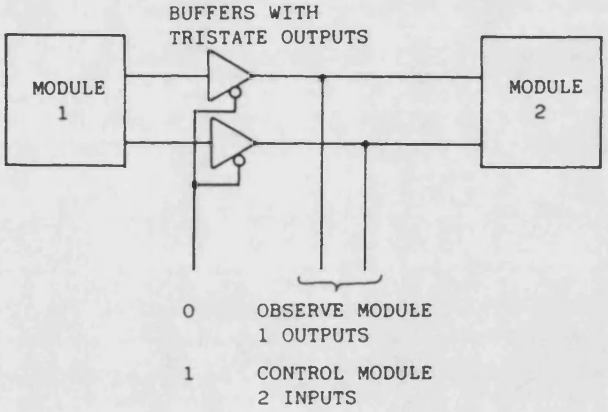


FIGURE 3.9 OBSERVATION & CONTROL OF MODULE INTERCONNECTIONS

CHAPTER FOUR : TECHNIQUES FOR BUILT IN TEST

Built in test (BIT) allows testing to be implemented in one of two modes, on or off line. Ideally it has negligible effect on system performance and involves minimal space and cost overheads. At the same time, it seeks to transform the fault diagnosis of a complex circuit from almost impossible to a relatively straight forward operation. It also needs to minimise test generation, test procedures and the assistance of external equipment.

BIT ranges from well established (and widely used) coding techniques to the more recent and powerful concept of a Serial Shadow Register. Both of these are described in some detail in this Chapter, along with three more important forms of BIT; Scan Design, the Built In Logic Block Observer and Autonomous testing. Many other techniques for BIT have been proposed, so a further selection are concisely detailed.

It is interesting to see what impact these techniques have had on chip manufacturers. A number of these techniques are being implemented at chip level and a summary is provided of some of the available devices.

However, prior to any of this discussion, section 4.1.2 presents the Linear Feedback Shift Register (LFSR), a circuit which is extensively used for data generation and compression.

The chapter concludes with a generalised approach to BIT, the concept of self verification.

Many of the BIT techniques presented can either individually, or in combination make extensive use of self checking circuits at chip, board or system level, as they do in the error detection processor [4.1,4.2]. These aspects are not covered here, but dealt with in subsequent

Chapters on self checking circuits and systems.

4.1 : MODES OF TESTING

Testing or checking can be implemented concurrently with normal operation (on line), or distinct from normal operation as an off line process, using the circuitry as it is, or in a special test mode of operation [4.16].

Fig. 4.1 shows the on and off line techniques for BIT considered in subsequent sections, together with their interactions. It also includes, for completeness, certain referenced aspects of on line testing for fault tolerance.

4.1.1 : On Line Testing

The need for on line testing originates from the requirements of fault tolerant computing, where ideally there is no computer down time due to failure. This is certainly true for critical space and aviation applications where faults must be detected, located and either the erroneous data corrected, or the faulty module replaced with built in spares.

Concurrent checking therefore requires, in general, a significant hardware redundancy to achieve its goal with zero time overhead. This method of testing also has the ability to detect intermittent errors that occur during normal operation, which an off line test procedure may not detect.

Coding schemes or duplication and comparison are the most common forms of concurrent testing.

4.1.2 : Off Line Testing

Off line testing uses hardware and/or software to verify that the operation of devices assembled as part of a system or in isolation is correct. The only restriction

on testing time is that it falls within an acceptable limit.

Non concurrent testing allows all modules to be thoroughly exercised, a process which can locate failures not discovered during concurrent testing methods because normal operation does not create such an extensive stimulus.

Increased observability and controllability (see section 3.2) is often created by the test mode of operation whereby certain circuitry, particularly sequential logic, is reconfigured. Scan design is one such technique.

4.2 : THE LINEAR FEEDBACK SHIFT REGISTER

Mechanisms for generating a circuit stimulus and observing its response are essential to the success of any BIT technique. These are provided by effective and compact circuits for test pattern generation and data compression respectively. The most widely used device to provide both is the linear feedback shift register (LFSR), particularly in scan path design.

A series of D-type flip-flops configured with feedback paths, as shown in Fig. 4.2a, is known as a LFSR and can be used to provide a pseudorandom pattern generator. When the feedback paths are carefully selected, it can generate a long, near random, sequence of output patterns with length 2^N-1, where N is the number of flip-flops in the shift register. Fig. 4.2b gives these patterns for the depicted LFSR. The philosophy of pseudorandom patterns is that by applying a sufficiently large number of them to the circuit, acceptable testing will be achieved. The LFSR allows test patterns to be applied at high speed, so that several million can be achieved within an acceptable test time.

A relatively short test sequence can produce a considerable amount of response data, since the state of each

output must be checked after the application of each stimulus. Data compression is therefore essential and can be achieved with the use of a LFSR plus an external input. This technique has already been covered in section 2.1, as it is signature analysis. Fig. 4.3 shows a four stage LFSR for this purpose and the resultant signature from a sample data stream. However, since most circuits have more than one output, and to avoid having a separate LFSR for each of these, the basic LFSR can be expanded to accommodate more than one input, as shown in Fig. 4.4. This is then known as a multiple input signature register (MISR) and is obviously more economical than the basic LFSR, but can suffer from a worse error detection probability [4.31].

4.3 : ERROR DETECTING CODES

The most common form of BIT is coding. It provides an on line (concurrent) method of verifying circuit operation, using both information and hardware redundancy.

A set of lines, typically a bus, are encoded, ideally at their source, and the code subsequently checked at various points in the circuit, as shown in Fig. 4.5. The binary combinations on these lines will then only be a subset of all possible combinations during normal (fault free) operation. A failure in this or associated circuitry will (ideally) produce a value outside this allowed subset and will therefore be detected by the checking process.

This can be expressed more formally as follows [4.32]. For a particular code, the allowed binary combinations or vectors are a subset S of a universe U chosen so that a likely failure affecting vector X_i in S produces vector X_i' which is not in S. A codeword is a vector in S and a a non codeword a vector in the set U-S. If X_i is a codeword and X_i' a different vector produced by a failure f, then f is a detectable error if X_i' is a non codeword and f an undetectable error if X_i' is the same or another

codeword. These concepts are illustrated in Fig. 4.6.

The choice of code is dependent on the following:

1) The types of fault to be detected.
2) Whether logical or arithmetic processing is involved, since some codes are not preserved during these operations.
3) The added complexity to the uncoded circuit.
4) The cost of installation.
5) How easily the code is checked.

Codes essentially fall into two categories, separable and non separable. In a separable (systematic) code, the codeword is formed by the concatenation of additional (separate) check bits to the original data word. In a non separable (non systematic) code, the original data word is extended and modified to form the codeword.

4.3.1 : Single Parity Codes

Parity is one of the simplest coding techniques available and has been extensively used for many decades. In its basic form, a single parity bit is appended to an n-bit data word to produce odd or even parity. For even parity the parity bit is a '1' or a '0' such that the total number of ones in the word (including the parity bit) is an even number, whilst for odd parity the total number of ones is odd.

Expressed mathematically the parity bit P for an n-bit word $x_1 \ldots x_n$ is:

$$\text{Even parity: } P_{even} = x_1 \oplus x_2 \oplus \ldots \oplus x_{n-1} \oplus x_n \qquad (4.1)$$

$$\text{Odd parity: } P_{odd} = \overline{x_1 \oplus x_2 \oplus \ldots \oplus x_{n-1} \oplus x_n} \qquad (4.2)$$

This is illustrated in Fig. 4.7.

Considering odd parity, any fault which changes the number of ones from odd to even is detectable. Thus odd parity will detect all single bit errors as well as multiple bit errors which are odd in number. The same is true for even parity.

The code is checked with an Exclusive-OR (EXOR) tree, as shown in Fig. 4.8. An MSI package is available for this purpose [4.33].

Parity is used extensively for checking data transmission paths and memory, but is not, in general, preserved by arithmetic or logical operations, as indicated in Fig. 4.9. If parity is to be used with arithmetic or logical operations, then the parity bit can be regenerated after the operation with the use of parity prediction techniques [4.34-4.36].

Overall parity creates the least redundancy (one bit) and is the cheapest code to generate and check.

4.3.2 : B-Adjacent Codes [4.37,4.38]

Basic parity is ideally suited for memory where each device handles a single bit, for example, 1K words x 1-bit RAM chips, since it will detect all failures within that device. However, if the memory device is four bits wide (for example, 1K words x 4-bits) then an internal fault could cause a four bit error, which is undetectable by simple parity. In this case the b-adjacent code can be used. This is illustrated in Fig. 4.10, where the technique is also known as interleaved parity. An error in b adjacent bits is only a single error for each of the b parity check codes. The checker now consists of b x (k+1)-input EXOR trees and one b-bit AND gate.

4.3.3 : Duplication Codes

Where it is difficult to code the internal paths of a

circuit or device, then the easiest solution is to duplicate the whole circuit and compare the outputs of normal and duplicated modules to detect failures in each, as shown in Fig 4.11.

The advantage of this technique is that all failures (single and multiple) will be detected, except those which identically change the outputs of both modules. To overcome the possible occurrence of identical failures occurring in each module, Sedmak [4.14,4.39] suggests that the duplicated module should be an inverse of the main circuit.

Duplication is a method of coding because when corresponding outputs are compared, the input to the comparator for each pair will be ‹00› or ‹11› for modules giving identical outputs, the duplication code, and ‹01› or ‹10› for modules with complementary outputs, the 1-out-of-2 code. Thus errors are indicated by an input pair of ‹01› or ‹10› for the duplication code and ‹00› or ‹11› for the 1-out-of-2 code. The 1-out-of-2 code is preferred since it detects unidirectional multiple errors, such as those caused by a loss of power.

Duplication provides the highest fault coverage, but has the most redundancy and is the most expensive form of checking. Larsen [4.40] compares the redundancy of duplication with the redundancy of other coding forms.

4.3.4 : Checksum Codes [4.41]

Checksum codes have been used extensively to detect errors in data transmission and data storage. Data is stored and transmitted in b-bit words. Appended to the end of each packet is a check word or checksum which is the binary sum of the words within that packet. The length of these packets may be anything from one to thousands of bytes.

The principle method of checking the code is software simply because of the vast amount of data involved. However, hardware detection of errors in individual words may be performed by breaking the word down into k x b-bit bytes and applying the technique to these bytes, as shown in Fig 4.12.

An extra b-bit byte is required for the checksum which will detect all faults within a single byte. The checker consists of (k-1) x b-bit binary adders and a b-bit comparator.

Checksums can also be incorporated into arithmetic operations with the use of checksum prediction [4.42].

Checksums provide the same fault coverage as the b-adjacent code in Fig. 4.10, but with a checker which is more expensive and slower.

4.3.5 : AN Codes [4.43]

The AN code is an arithmetic code [4.44,4.45], that is to say, unlike parity, is preserved during arithmetic operations. The code is described as non systematic and non separable.

In the AN code an uncoded word X is multiplied by a check base A to form the codeword AX. Each codeword is formed by appending $\lceil \log_2 A \rceil$ bits to X. Binary addition of codewords is performed so that:

$$AX + AY = A(X+Y) \qquad (4.3)$$

Thus the encoding of the sum of two numbers is the sum of their respective encodings. When no fault occurs, the adder output is a multiple of A. A checker monitors this output and produces an error signal when the residue or remainder is not zero. This structure is illustrated in Fig. 4.13, together with a numerical example. All faults

producing an output which is not a multiple of A will be detected. These faults include most unidirectional multiple errors.

In this general case, the checker must carry out long division by A to obtain the residue which is a slow and complex hardware algorithm. However, this process can be greatly simplified if the check base $A = 2^b-1$, where b is an integer greater than one [4.46]. This is known as the low cost AN code because the checker now greatly simplifies to an adder tree of b-bit modulo 2^b-1 adders plus an AND gate. A sample structure together with a numerical example is given in Fig. 4.14. Note that:

i) The modulo 2^b-1 adders are ordinary binary adders with end around carry.
ii) The correct residue is now all 1's (all 0's would be produced by normal long division), so the final check of the residue might need to take into account all 0's or all 1's as being correct.

All errors within a b-bit byte can be detected except those of magnitude 2^b-1. This form of coding has similar redundancy and checker costs to the checksum code, but may be a little slower. The AN code is not suitable for logical operations since it is non separable.

4.3.6 : Residue Codes [4.47]

The basic residue code is, again, an arithmetic code and similar to the AN code except that it is separable. A word X is encoded by appending check bits C(X) to it which are the residue of X divided by a check base A. The number of check bits required is given by $\lceil \log_2 A \rceil$. The encoding of the sum of two numbers is the binary sum of the original data words, say X and Y, and the independent summation of their respective check bits modulo A. The encoding of the sum of X and Y, f(X+Y), is then:

$$f(X+Y) = \langle C(X) +_A C(Y),\ X+Y \rangle \qquad (4.4)$$

where $+_A$ is addition modulo A and ',' is concatenation.

A checker which compares C(X+Y) with $(C(X) +_A C(Y))$ will then be used to detect errors in the addition, as shown in Fig. 4.15. All faults will be detected except those which produce an all 0's or all 1's residue.

As with the AN code, a low cost residue code can be produced by using $A = 2^b-1$, where b is again an integer greater than one. This is now beneficial to both the generation and checking of the code. A sample structure and numerical example are given in Fig. 4.16. The same fault detection capabiltiy, redundancy and checking costs apply as for the AN code.

4.3.7 : M-out-of-n Codes [4.48]

M-out-of-n codewords are a subset of all n-bit words where exactly m of its bits are a logic 1. The code is therefore termed a fixed weight code. The number of possible combinations for m bits out of an n-bit word is given by:

$$C_m^n = n!/m!(n-m)! \qquad (4.5)$$

Although m-out-of-n codes are non separable and non systematic, and therefore not suitable for processing systematic data, they are suitable for encoding control information, where the individual bits of a word have specific functions rather than a numerical representation. In some cases, for example, memory address decoding, they occur naturally. They can detect all unidirectional multiple errors, except faults which change one codeword to another codeword.

A maximum number of codewords occurs when m equals the integer part of n/2 : $m = \lfloor n/2 \rfloor$. Using this particular

case and an 8-bit data word, consider the following. If the word is encoded using simple parity, then $2^8 = 256$ codewords are provided with one redundant bit. A similar number of codewords (252) are available from a 5-out-of-10 code which requires two redundant bits. So, compared with parity, n/2-out-of-n codes, better known as k-out-of-2k codes, require slightly more redundancy for much better fault coverage.

4.3.8 : Berger Codes [4.49-4.52]

Berger codes are optimal separable (systematic) codes, where an n-bit word X is encoded by the addition of k checkbits. The checkbits are the binary representation of the number of 0's in X, X0, or the binary representation of the number of 1's in X complemented (bit by bit), X1. This results in k check bits, where $k = \lceil \log_2 (n+1) \rceil$, the number of binary bits needed to represent n. X0 and X1 will only be identical when $n = 2^k - 1$, when a maximal length Berger code is produced.

Berger codes will detect all unidirectional errors and are optimal, in terms of the number of check bits required for n information bits, amongst all separable codes that detect unidirectional errors. However, they are more redundant than m-out-of-n codes, which also detect unidirectional errors, but Berger codes have the separability advantage which simplifies and minimises encoding and decoding hardware.

The checker consists of a series of full adder modules to add the information bits in parallel. Fig. 4.17 shows a structure for n=7 and k=3, together with a numerical example.

4.3.9 : Cyclic Codes [4.53]

Binary cyclic codes have the property that any cyclic shift of a code word is also a code word. The encoding

and decoding processes for these codes are straightforward with the use of linear feedback shift registers (see section 4.2), so the codes are ideal for checking serial data.

An n-bit binary number may be represented as a polynomial M(x) thus:

$$M(x) = m_{n-1}x^{n-1} + m_{n-2}x^{n-2} + ... + m_1x + m_0 \qquad (4.6)$$

Two forms of cyclic code exist; systematic or non systematic.

4.3.9.1 : Non Systematic Cyclic Codes

A code polynomial V(x) is formed from the product G(x)M(x)

$$V(x) = G(x)M(x) \qquad (4.7)$$

where G(x) is a suitable generator polynomial of the form

$$G(x) = g_rx^r + g_{r-1}x^{r-1} + ... + g_1x + g_0 \qquad (4.8)$$

For an k-bit encoded word r = k-n (n is the number of information bits).

This code is non systematic since the information bits are scrambled in the polynomial product. Fig. 4.18 gives an example.

4.3.9.2 : Systematic Cyclic Codes

V(x) is formed by appending the inverse of the remainder when $x^rM(x)$ is divided by G(x) as check bits to the original information bits.

$$x^rM(x)/G(x) = Q(x) + C(x)/G(x) \qquad (4.9)$$

$$\text{i.e. } x^rM(x) = Q(x)G(x) + C(x) \qquad (4.10)$$

$$V(x) = x^r M(x) - C(x) = Q(x)G(x) \qquad (4.11)$$

where: r = k-n as above.
Q(x) and C(x) are polynomials, with the highest power of x in C(x) being less than r.

Thus V(x) is a multiple of G(x). Fig. 4.19 gives an example.

These cyclic codes are referred to as (k,n) codes. They will detect all single errors and all burst errors that affect k-n (r) or fewer adjacent bits. The cyclic codes can also be used for error correction.

Both the encoding and decoding of systematic cyclic codes requires the use of division by G(x):

Encoding: $x^r M(x)/G(x)$ and use the inverse of the remainder as check bits.

Decoding: V(x)/G(x) and if the remainder is zero it is a codeword.

A linear feedback shift register (LFSR) can be used to serially divide by G(x). Fig. 4.20 shows such a structure where G(x) is that used in the above example.

4.3.10 : Hamming Codes [4.12,4.54]

The Hamming codes extend the principles of single parity. They are not only able to detect errors, but also able to correct them as well. In these codes more than one parity bit is appended to a data word X, such that each added bit represents the parity of a different subset of the n information bits.

Codes are e-error detecting if any fault causing up to e erroneous bits can be detected and f-error correcting if

f erroneous bits can be corrected to produce the original fault free data.

The Hamming distance d of a code is the minimum number of bits between two codewords. Fig. 4.21 shows the interrelation of d,e and f derived from coding theory [4.12]. For simple parity d=2, which allows single error detection and zero error correction. Since the Hamming codes provide error correction, they must have d greater than 2.

Consider the case for a single error correcting code [4.12]. To be single error correcting a data word of n information bits requires c check bits, where

$$2^c \geq n+c+1. \qquad (4.12)$$

For an overall b-bit word, the check bits ideally occupy positions b_{2^0}, b_{2^1}, b_{2^2}, ..., $b_{2^{c-1}}$.

As an example, let n=4 such that (4.12) gives $c \geq 3$. Take c=3 and a seven bit word is required, b_7 .... b_1, with the check bits occupying b_1, $b_2$and b_4.

The values of the check bits are determined from the parity check equations:

b_4 such that : $P3 = b_7 \oplus b_6 \oplus b_5 \oplus b_4 = 0$ (4.13)

b_2 such that : $P2 = b_7 \oplus b_6 \oplus b_3 \oplus b_2 = 0$ (4.14)

b_1 such that : $P1 = b_7 \oplus b_5 \oplus b_3 \oplus b_1 = 0$ (4.15)

These are even parity equations, so an even number of 1's is required to satisfy the left hand side of each equation.

If the information bits have values b_3=1, b_5=0, b_6=0 and b_7=1, then the check bits have values b_1=0, b_2=0 and b_4=1, so that the encoded word is 1001100.

If a single bit error occurs in bit b_5 the word becomes

1011100. When checking occurs, P3=1, P2=0 and P1=0. It can be seen from the parity check equations that if the outputs P3, P2 and P1 are concatenated in this order, then they represent the binary number of the erroneous bit, in this case 101, i.e. b_5, which can be corrected.

Consider now a fault in bits b_5 and b_2, i.e. a double fault. The modified word becomes 1011110, with P1, P2 and P3 all equal to 1. Accordingly, bit 7 will be corrected producing 0011110. This demonstrates that although the double bit error has been detected, it cannot be corrected. In fact, an incorrect correction will be made. Fig. 4.22 shows pictorially why this occurs.

A more typical information word of eight bits requires at least four check bits, from (4.12). In this case, it is equally redundant as a Berger code with n=8, but has twice as many redundant bits for n=8 in an equivalent k-out-of-2k code (5-out-of-10). However, whilst the k-out-of-2k code can detect all unidirectional errors it has no error correction capacity (d_{min}= 2), whereas the Hamming code considered has single error correction and double error detection.

The checking requirement for this Hamming code is c parity trees, each with up to 2^{c-1} inputs (c is the number of check bits).

Practical implementation of the Hamming code in microprocessor based systems is presented by Heimlich [4.55] and Wall [4.13].

4.4 : SCAN DESIGN

Scan design is a technique developed by IBM which involves the reconfiguration of storage elements. It exists in two main forms; Scan Path design using conventional storage elements and Level Sensitive Scan design using special storage elements. 'STUMPS' extends the scan concepts for

multi-device applications.

4.4.1 : Scan Path Design [4.31,4.56-4.58]

In scan path design all storage elements (flip-flops) have additional circuitry as indicated in Fig. 4.23. This additional circuitry is essentially a 2 to 1 multiplexer in front of the D input, with normal data and scan data as the two inputs.

During test, this circuitry enables the storage elements to be reconfigured serially into a shift register, henceforth termed the scan register, as shown in Fig. 4.24. The scan data out from one storage element is inherently connected to the scan data input of the next storage element and it is this data path which is selected for test purposes by the scan enable line. A sequential circuit is then reduced to a shift register and combinational logic, so that test patterns are required for the combinational logic only (a comparatively easy task compared with test generation for sequential logic). These are applied using the normal input pins and the access path provided by the scan register. If, for example, a LFSR data generator and LFSR data compressor are added with appropriate control circuitry, then the test setup becomes that in Fig. 4.25.

For Fig. 4.25 testing is as follows:

1) Reset the data generator and the data compressor and then enable the data generator.
2) Set mode to test. This reconfigures the storage elements into a scan register.
3) Clock the circuit until the scan register has been completely loaded with data (via scan in). This is the first test for the combinational logic.
4) Set mode to normal and set up primary input data.
5) Clock the circuit once.
6) Set mode to test, latching primary output data.

7) Enable the data compressor (first time only) and shift out the contents of the scan register into it, whilst simultaneously loading the next test pattern.
8) Repeat steps 4)-7) until the test generator has supplied all possible test patterns.
9) Inhibit data generator and data compressor.
10) Compare the signature held in the data compressor with a reference value and generate a pass/fail indication.

4.4.2 : Level Sensitive Scan Design [4.58-4.63]

The level sensitive scan design (LSSD) technique is similar to the scan path, except that the storage element is a shift register latch (SRL) which uses level sensitive latches in a master-slave configuration, as shown in Fig. 4.26. This circuit is nearly independent of hard to control ac characteristics such as rise time, fall time and overall delay for correct operation and is hazard/race free. However, there are restrictions on when and how signals can change [4.59].

The scan data out is now a separate line, but still connects to the scan data in of the next unit to form the scan register, which is now operated from a two phase non-overlapping clock [4.59-4.61].

LSSD is used extensively in the IBM System/38 [4.60,4.61].

4.4.3 : Advantages and Limitations of Scan Design

Advantages:

1) Sequential logic is transformed into a combinational form, allowing well understood test generation techniques such as the D-Algorithm or Boolean Difference [4.64,4.65] to be used.
2) It provides a vast improvement on the controllability and observability of internal nodes compared to a circuit without this technique.

3) It can be used for chip, board and system testing during production and in the field.
4) The additional circuitry creates little performance degradation during normal operation.

Limitations:

1) Each storage element is two to three times logically more complex than a standard unit (although the additional latches required for LSSD can be used for other system functions).
2) Up to four more I/O pins are required.
3) External control and clock circuitry is required.
4) External inputs will have to be precisely controlled to generate the correct response (signature).
5) It provides only a dc test of the system.
6) The serial nature of the data can result in long test times.

Scan path can be added to conventional designs [4.57], whilst LSSD is included during design.

The above assumes that all the storage elements in a circuit will be serially connected to form a shift register, giving a complete scan path, but Trischler [4.66] considers the effect of not including every element, which results in an incomplete scan path.

4.4.4 : STUMPS

The concepts discussed above may be used for circuits within a single device or several devices cascaded. However, for multiple device applications, self test using MISR and a parallel shift register sequence generator (STUMPS) is an extension of these ideas, as shown in Fig. 4.27 [4.31].

The scan inputs are fed from the parallel output of a LFSR pseudorandom sequence generator, whilst the scan outputs

feed the parallel inputs of a MISR via a set of control gates. The control gates allow the test response from a device or devices to be ignored in order to isolate an incorrect signature to one particular chip. Obviously additional known signatures will be required for this purpose.

Using this technique, a long shift register has been broken down into a number of smaller units operating in parallel, so the test time will be significantly reduced.

4.5 : THE BUILT IN LOGIC BLOCK OBSERVER [4.58,4.62, 4.67-4.69]

THE Built In Logic Block Observer (BILBO) combines many of the elements used in scan design to provide a self testing structure. In a similar manner to the scan path technique, the BILBO is formed from reconfigured storage elements within a network, plus a number of additional gates.

It consists of latches with feedback paths via EXOR gates, which are typical of LFSRs. Fig. 4.28a shows a typical BILBO structure which has parallel inputs from combinational logic Zi, parallel outputs Qi,a serial input SDI, a serial ouput SDO and two control inputs B1 and B2.

B1 and B2 select four operating modes:

Mode 1 : B1=B2=1 - Fig. 4.28b

This is the normal mode of operation. The circuit operates as a parallel latch with data loaded at the Z inputs on a transition of the clock.

Mode 2 : B1=B2=0 - Fig. 4.28c

The circuit becomes a shift register. Data present at SDI will be shifted through the four latches, appearing at Zi and SDO.

Note that the combination of Modes 1 and 2 provide a scan register.

Mode 3 : B1=1, B2=0 - Fig. 4.28d

The circuit now becomes a MISR. It can be used to compress test responses or, for constant Zi, generate pseudorandom data streams at Qi.

Mode 4 : B1=0, B2=1 - Fig. 4.28e

On a clock transition all latches will be reset (0 applied to all inputs).

The BILBO is thus an incredibly versatile building block for the construction of self testing circuits, as indicated in Fig 4.29. Alternatively, it can be used to assist other BIT techniques. The scan path structure of Fig 4.25, for example, can be modified to include a BILBO which will perform the data generation and data compression functions simultaneously, such that the response of the circuit to one test influences the next test, as shown in Fig 4.30. Once a failure has been detected, the other two modes of operation allow the circuit to perform as a conventional scan register for further diagnosis.

Fasang and Konemann [4.67-4.69] detail experimental results and usage for the BILBO, allowing BIT at normal system speed and the high fault resolution quality of scan path techniques.

4.6 : AUTONOMOUS TESTING [4.62,4.70-4.74]

Scan design provides a powerful testing mechanism, but introduces additional complexity into the design process and requires test pattern generation (TPG) [4.64,4.65] determined by the type of faults to be detected (fault modelling). Scanning patterns in and out of the network means a long testing time, especially for large circuits. Even if pseudorandom patterns are used to eliminate TPG (as prevously described), then the test time will be substantially increased, since there must be sufficient patterns to provide the desired fault coverage.

Autonomous testing does not require fault modelling or TPG. All possible input combinations are applied to combinational logic and all sequences to sequential logic, with outputs monitored for correct operation. This is known as exhaustive testing, which, in general, is a lengthy process. However, the testing time can be substantially reduced by partitioning the circuit under test into subcircuits which have a relatively small number of inputs compared to those of the complete circuit. Williams [4.62,4.63] indicates that testing time is related to the cube of the complexity (number of gates) in the circuit.

Requirements for this technique are:

1) Mechanisms to partition the circuit under test into subcircuits.
2) Additional circuitry, or reconfiguration of existing circuitry, to provide all input combinations to the subcircuit.
3) Additional circuitry, or reconfiguration of existing circuitry, to verify the response of the subcircuit.

As with many of the BIT techniques, the circuit functions in either a normal or a test mode of operation. The test mode of operation is as described above. All additional test circuitry is transparent to the inputs and outputs during the normal mode of operation. The transformation from the normal to the test mode of operation is presented in Fig. 4.31.

Parts 2) and 3) of the strategy above are implemented with the now familiar LFSR, modified so that the generator can provide all output combinations, since the LFSR in Fig. 4.2 does not generate the all zero state. A circuit proposed by McCluskey (MC) in [4.73], and given in Fig. 4.32, closely resembles the BILBO already described.

In order to exhaustively test each subcircuit, the partitioning must allow all its inputs to be controllable and all its outputs observable.

Two approaches are presented for partitioning by MC [4.73]. These are the use of multiplexers and sensitised partitioning.

4.6.1 : Partitioning with multiplexers

The use of multiplexers is demonstrated in Fig. 4.33. It shows a module G broken down into two subcircuits G1 and G2 (Fig. 4.33a); how multiplexers are added to create the subcircuits (Fig. 4.33b); how the multiplexers are configured for normal operation (Fig. 4.33c) and how the multiplexers are configured to test subcircuit G1 (Fig. 4.33d). Bozorgui-Nesbat (BN) [4.72] indicates that a stuck-at fault on any line, including the testing circuitry, will be detected, whilst MC [4.73] warns against failures affecting normal operation only. BN [4.72] also details a generalised approach to partitioning in this manner. MC [4.73] describes a practical implementation of this technique for a 74181 ALU/Function Generator [4.69] and compares the number of tests required with and without the partitioning. Buehler [4.76] carries out a similar operation for a full/half adder combination.

However, this method of partitioning involves an additional overhead to provide the multiplexers and their control circuitry. This overhead is dependent on the size and complexity of the network and could be significant. The multiplexers also reduce the speed of normal operation.

4.6.2 : Sensitised Partitioning

MC [4.73] suggests that the effect of multiplexer insertion should be created using sensitised partitioning. Here circuit partitioning and subcircuit isolation are achieved by applying an appropriate pattern to some of the

input lines. As an example, consider the circuit G in Fig. 4.34a divided into the two subcircuits G1 and G2 of Fig 4.34b. These subcircuits can be tested as follows:

1) To test G1 : Put D1=D2=0, then E1=C1 and E2=C2 (outputs of G2=outputs of G1). Apply all binary combinations to {A1,A2,B1,B2}.

2) To test G2 : Put B1=B2=1, then C1=A1 and C2=B2 (inputs to G2=inputs to G1). Apply all binary combinations to {A1,A2,C1,C2}.

The test configurations for G1 and G2 are shown in Figs. 4.34c and 4.34d respectively. Each subcircuit effectively has four inputs and therefore requires 2^4=16 test patterns, so a total of 32 tests are required to test both subcircuits, compared with 2^6=64 if the circuit is tested as a single unit. MC [4.73] again applies the technique to the 74181 ALU/Function Generator. In addition, MC suggests how to implement the pattern generator and response verifier within the same device.

Unlike the insertion of multiplexers, sensitised partitioning is heavily dependent on circuit structure, but requires less additional hardware overhead.

MC [4.70] extends the above principles for autonomous test to sequential circuits.

4.7 : SERIAL SHADOW REGISTER [4.58]

Serial shadow register (SSR) diagnostics is an on-chip technique which allows complex systems to be easily controlled and observed.

The structure, shown in Fig. 4.35, uses 'output' and 'shadow' registers to initialise and test combinational and sequential circuits. External control circuitry allows the data transfer between the shadow register and

the circuit under test to be bidirectional. Fig. 4.36 details the various operations of the SSR structure.

The output and shadow registers are both connected to the D and B buses. The shadow register has either serial or parallel input data. Serial data arises from test patterns shifted in, or test responses shifted out, i.e. scan data as in the scan path technique, whilst parallel data is loaded from either the B bus or the output register. A 2 to 1 multiplexer allows the output register to be loaded from either the D bus or the shadow register. It can thus obtain serially input test patterns from the shadow register, or have its contents (test responses) copied to the shadow register to be serially output. Normal operation can continue immediately after system states have been read from the output register, unlike the Scan Path and BILBO techniques, in which these states have to be reloaded as they are destroyed by the read process. In addition, switching to the test mode does not alter system states in the output register, this only happens when the test pattern is loaded.

DCLK and CLK, generated from the system clock, clock the shadow and output registers respectively, whilst the mode and SDI inputs control data loaded into the registers from external sources. Together these four lines control the on-chip diagnostics, as detailed in Fig. 4.36. The two clocks increase test speed since test patterns can be loaded (or test reponses unloaded) during normal output register operation.

Fig 4.37 shows a typical application of the SSR technique, where SSRs replace all ordinary and I/O registers.

Overall, the technique allows a great deal of flexibility in the test process and although it requires more hardware than other techniques (shadow registers, mutilplexers and control circuitry),its superiority is highlighted by a comparison with the Scan Path and BILBO structures [4.58].

4.8 : OTHER BIT TECHNIQUES

The five techniques considered so far are the most widely used forms of BIT. However, there are a number of other techniques which will be briefly mentioned for completeness.

4.8.1 : Scan Set Logic

A technique proposed by Sperry Univac which combines the principle of scan path and SSR to provide a scan register independent of system latches and thus not part of any system data path [4.62].

4.8.2 : Random Access Scan

Random access scan is similar to the scan path technique except that shift registers are not employed. Instead, an addressing scheme allows each latch to be selected for control and observation purposes [4.62].

4.8.3 : Syndrome Testing

All 2^n patterns are applied to an n-input combinational circuit and its syndrome compiled [4.62,4.77]. The syndrome is essentially the number of 1's appearing at the output of the circuit during the test, which is then compared with its correct value to locate a faulty network. However, some circuits need to be modified so that they are syndrome testable for stuck-at faults. Savir [4.77] demonstrates this technique for the 74181 ALU.

4.8.4 : Walsh Coefficient Testing

Again, this technique requires the application of all possible input patterns to a combinational network, in which two of its Walsh coefficients are checked to de-

termine the presence of stuck-at faults [4.62,4.78-4.80]. Modification of the network might also be neccessary so that it is testable by this method.

4.8.5 : ROM Based Test Patterns and Responses

Instead of using test patterns from an external source or a LFSR, patterns are stored in a special test ROM [4.81]. Test responses are compared with expected responses stored in another ROM.

4.8.6 : Self Oscillation

During its test mode of operation, the circuit is reconfigured so that it will oscillate only if it is fault free [4.76]. Specific input combinations are required to create this condition.

4.8.7 : Self Comparison

In a self comparison scheme, the circuit under test is partitioned into two subcircuits which produce identical outputs for a given set of inputs [4.76,4.81]. Checking then simply involves a comparison of the corresponding outputs in each subcircuit.

4.8.8 : History Memory

Not strictly a BIT technique perhaps, but worth a mention as it is ideally suited for computer applications [4.82]. A large quantity of memory records particular or all bus transactions for subsequent analysis, so that hardware and software failures can be located. In principle, it is similar to a logic state analyser.

4.8.9 : Reed Muller Canonical Representation

An n-input combinational circuit can be implemented in Reed Muller canonical form thus [4.83]:

$$f(x_1,\ldots,x_n) = C_0 \oplus C_1x_1 \oplus C_2x_2 \oplus \ldots \oplus C_nx_n \oplus C_{n+1}x_1x_2 \oplus C_{n+2}x_1x_3 \oplus \ldots \oplus C_{2^n-1}x_1x_2\ldots x_n \quad (4.13)$$

This form requires, at most, (3n+4) tests to detect all single stuck-at faults, including input faults [4.83].

4.8.10 : Technique Comparisons

Overall, certain papers make useful comparisons between techniques in terms of additional hardware requirements, fault coverage, test time, pattern generation and test response mechanisms, testability of the test circuitry, VLSI applicability and so on.

Buehler [4,76] compares self oscillation, self comparison, scan path and BILBO; Bhavsar [4.81] compares duplication, residue coding, ROM based test patterns and responses, autonomous testing and self comparison; whilst Gupta [4.58] compares the SSR, BILBO and scan path techniques.

4.9 : INTEGRATED CIRCUITS WITH BIT

A number of integrated circuits which either contain BIT, or assist BIT, are already commercially available, with hopefully many more to come.

Some of these devices are reviewed below, with reference to their function and their BIT.

4.9.1 : Motorola MC6805 : single chip 8-bit microprocessor

The 6805 has 116 bytes of on-board ROM and control circuitry which allow a self check mode of operation [4.84-4.86]. When certain external requirements have been met, the self check mode of operation can be initiated. The internal program then runs repeatedly until it is manually terminated or it discovers a fault. It firstly

tests I/O lines on which a test status is emitted, proceeding to test the other I/O lines, RAM, ROM and interrupts.

4.9.2 : Motorola M68000 : 16-bit microprocessor

The 68000 does not strictly have BIT, but it does have exception processing - processing for exceptional conditions [4.87,4.88]. These exceptions can be generated internally or externally and include divide by zero, odd address accesses, illegal instructions, bus errors, privilege violations and spurious interrupts. The externally generated bus error can also cause the processor to re-run the bus cycle in which the error was detected. If a double bus error occurs during exception processing, then the processor is halted. Also, a hardware trace faciltity is provided which causes exception processing after each instruction is executed, allowing a debugging program to monitor main program execution.

4.9.3 : Intel iAPX 432 : 32-bit microprocessor

The iAPX 432 consists of a two-chip general data processor (GDP), interface processor, bus interface unit (BIU) and memory control unit [4.89,4.90]. It is aimed at multiprocessor applications and features a functional redundancy checking mode for hardware error detection. When the GDP devices are in a checking mode, they compare certain of their I/O lines with an internally generated result, signalling an error to the BIU. The BIU then serially transmits the type and location of the error to all other nodes in the system. Redundancy in this error reporting mechanism ensures that each node will receive the same correct error report. Each node logs the information and proceeds with an appropriate recovery procedure.

4.9.4 : Monolithic Memories 74S818 : Diagnostic Register

The 74S818 diagnostic register is a direct implementation

of the SSR technique and contains the circuitry detailed in Fig. 4.35 [4.91].

4.9.5 : Monolithic Memories 63DA1643/841/441 : PROMs

This diagnostic PROM family consists of devices with arrays of varying depths, but all 4-bits wide [4.91,4.92]. Each has a built in 4-bit SSR. Applications include a microprocessor control store with built in system diagnostic test, as shown in Fig. 4.37, serial character generation or serial code conversion.

4.9.6 : National Semiconductor SLX6360 : Logic Array

The SLX6360 is a 6000 gate semicustom logic array which has a BILBO [4.93]. A 189-bit pseudorandom number generator, which is externally seeded, can deliver 20M vectors per second. The test responses are fed to a checksum register which generates an 88-bit signature.

4.9.7 : National Semiconductor DP8400 : Error Corrector

The DP8400 provides error correction and detection within one chip [4.94]. Each chip handles 16 data bits, generates check bits using a modified Hamming code, indicates and corrects single and double bit errors and indicates triple bit errors. It can also differentiate between transient and permanent failures. In addition to all of this, it can generate and check single bit byte parity and allows access to all of its internal gates for diagnostic purposes.

4.9.8 : LSI Logic LSA2000 : Structured Array Family

In addition to logic gates, the LSA2000 family of structured arrays have various combinations of RAM, ROM and/or special purpose megacells [4.95]. They also contain multiple scan paths which allow every memory bit and every megacell transistor to be tested.

4.9.9 : Hitachi : Gate Arrays

Hitachi gate arrays also include multiple path scan design, referred to as a scan bus structure [4.96]. Master-slave latches are employed, similar to those in LSSD, although they are not level sensitive.

4.10 : SELF VERIFICATION

This section outlines the proposals presented by Sedmak [4.14,4.39,4.97] for self verification (SV). These proposals provide an excellent focus for all the various on or off line testing forms of BIT considered in previous sections, hence their inclusion.

Sedmak suggests a somewhat ideal, but generalised approach to BIT, which merges on and off line testing so that verification of a fault free chip, board or system can be made without the use of external test equipment and with logic operating at its normal speed.

The overall scheme is shown in Fig. 4.38 and consists of the following:

1) A combinational/sequential partition or subcircuit of the complete circuit.
2) Internal Stimuli Generators (ISGs) which provide stimuli to thoroughly exercise the logic and expose any fault. They do not use stored test patterns or require test pattern generation.
3) An ISG Supervisor (ISGS) which coordinates the operation of all ISGs.
4) Fault Detection Circuits (FDCs) which monitor intermediate as well as final outputs, so that intermittent or solid faults are immediately detected.
5) An Error Status Generator (ESG) which collects information from the FDCs and encodes it to provide error signals which indicate the presence and location

of failures.

Both the ISGs and ISGS operate during verification and not during normal operation. The FDCs and ESG operate continuously during normal operation and self verification.

In design for self verification the following need to be considered:

1) The size and complexity of each partition.
2) The quantity and position of FDCs and ISGs, versus the complexity and position of the ISGS and ESG.
3) Detailed designs for FDCs, ISGs, ISGS and ESG.

In addition, it is desirable to:

a) Minimise the SV cost overhead.
b) Minimise the degradation of performance during normal operation.
c) Maximise the speed of the SV process.
d) Maximise fault coverage during both modes of operation.

Using the constraints of a) to d), Sedmak [4.97] expands points 1) to 3), giving extensive guidelines for each (although little attention is given to the ESG). He also applies the design for verification process to a 32-bit ALU.

4.11 : CONCLUSIONS

Overall, it is considered by the author that the most important aspect of BIT and design for verification is the use of on line fault detection. If this forms the sole basis for testing and verification of microprocessor based systems then the following are achieved:

1) The system functions only in its normal mode of operation, so there is no special test mode and no

reconfiguration of circuits.

2) Solid and intermittent faults are detected immediately.
3) No external test equipment is required.
4) No TPG or stimulus is required, since the circuit is continuously supplying its own.

Faults that affect normal system operation need to be detected, located and cause actions such as:

a) Notification to the outside world.
b) Error correction.
c) System halt or shutdown.
d) Operation retry [4.14,4.29,4.30].

There also has to be mechan-isms for fault detection. These are often coding techniques, which include duplication. The fault detection circuits are then code checkers as previously described (section 4.3). A problem then arises as to what happens if a failure occurs in the checking circuit. Is there a checker checking the checker, another checker checking that checker and so on ad-infinitum? Fortunately, the answer is no and the problem is resolved with the use of self checking checkers. Despite the presence of an internal fault, these circuits either produce a correct output or indicate that the failure exists. They are the basis for the rest of the thesis.

4.12 : REFERENCES

4.1) A monolithic self-checking error detection processor - J. Chavade, M. Vergniault, P. Rousseau, Y. Crouzet, C. Landrault;1980 Test Conf.*;pp. 279-286.

4.2) Design specifications of a self-checking detection processor - Y. Crouzet, C. Landrault; FTCS-10*; pp. 275-277.

4.3) Real-time fault diagnosics multiple microprocessor systems - H. Fu, G. M. Flachs; 1981 IEEE Real Time Systems Symposium; December 1981; pp. 56-61.

4.4) Microprogrammed control and reliable design of small computers - G. D. Kraft, W. N. Toy; Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA; 1981; pp. 362-365.

4.5) Design of a microprogram control for a processor in an electronic switching system - T. F. Storey; Bell Syst. Tech. J.; February, 1976; pp. 183-232.

4.6) Error detecting codes, self checking circuits and applications - J. F. Wakerly; Elsevier - North Holland; New York; 1978; pp. 3-5.

4.7) Diagnosis and reliable design of digital systems - M. A. Breuer, A. D. Friedman; Pitman Publishing Ltd., London, 1977; pp. 274-276.

4.8) Fault Detection in digital circuits - A. D. Friedman, P. R. Menon; Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA; 1971; pp. 193-202.

4.9) Error correcting codes for reliable digital circuits - I. S. Reed; Infotech*; pp. 713-745.

4.10) Error-correcting codes with byte error-detection capability - C. Chen; IEEE Trans. Comput.; Vol. C-32, No. 7; July, 1983; pp. 615-621.

4.11) A new class of error-correcting/detecting codes for fault-tolerant computer applications - D. K. Pradhan; IEEE Trans. Comput.; Vol. C-29, No. 6; June, 1980; pp. 471-481.

4.12) Error detecting and error correcting codes - R. W. Hamming; Bell Syst. Tech. J.; Vol. 29, No. 1; January, 1950; pp. 147-160.

4.13) Applying the Hamming code to microprocessor based systems - E. L. Wall; Electronics; November 22, 1979; pp. 103-110.

4.14) Fault-tolerance of a general purpose computer implemented by very large scale integration - R. M. Sedmak, H.L. Libergot; FTCS-8*; pp.137-143.

4.15) Designing reliable computer systems the fault tolerant approach - 1 - R. G. Bennetts; IEE Electronics and Power; November/December, 1978; pp. 846-851.

4.16) Designing reliable computer systems the fault tolerant approach - 2 - R. G. Bennetts; IEE Electronics and Power; January 1979; pp. 51-56.

4.17) Fault tolerance and digital systems - R. G. Bennetts; Microprocessors and Microsystems; Vol. 3, No. 8; October, 1979; pp. 365-373.

4.18) Fault tolerant computing: Techniques and development - A. Avizienis; Infotech*; pp. 309-333.

4.19) Architectural design for near - 100% fault coverage - J. J. Stiffler; FTCS-6*; pp. 134-137.

4.20) A study of standard building blocks for the design of fault-tolerant distributed systems - D. A. Rennels, A. Avizienis, M. Ercegovac; FTCS-8*; pp. 144-149.

4.21) The STAR (self-testing and repairing) computer: An investigation of the theory and practice of fault-tolerant computer design - A. Avizienis, G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, D. K. Rubin; IEEE Trans. Comput.; Vol. C-20, No. 11; November, 1971; pp. 1312-1321.

4.22) Design techniques for reliable hardware - F. P. Maison; Infotech*; pp. 236-251.

4.23) Fault diagnosis in computer control systems - G. Edge; Systems Technology; No. 31; April, 1979; pp. 33-41.

4.24) Availability, reliability and maintainability aspects of the Sperry Univac 1100/60 - L. A. Boone, H. L. Liebergot, R. M. Sedmak; FTCS-10*; pp. 3-8.

4.25) Installability, availablity, reliability and maintainability aspects of the Sperry System 11 - A. K. Bhatt, D. R. Mueller; FTCS-14*; pp. 29-35.

4.26) Dependable computing and fault tolerance: Concepts and terminology - J. Laprie; FTCS-15*; pp. 2-11.

4.27) Computer systems reliability: An overview - A. Avizienis; Infotech*; pp. 216-233.

4.28) Error detection process - model, design, and its impact on computer performance - K. G. Shin, Y. Lee; IEEE Trans. Comput.; Vol. C-33, No. 6; June, 1984; pp. 529-540.

4.29) On-line bus fault diagnosis in microprocessor systems - D. P. Agrawal, V. K. Agarwal; J. of Dig. Syst.; Vol. 4, No. 4; Winter, 1980; Computer Science Press Inc., USA; pp. 377-391.

4.30) Error correction by alternate-data retry - J. F. Shedlestky; IEEE Trans. Comput.; Vol. 27, No. 2; 1978; pp. 106-112.

4.31) Built-in test: A review - C. Maunder; IEE Electronics and Power; March, 1985; pp. 204-208.

4.32) as per 4.6) p. 9.

4.33) The TTL book for design engineers - The engineering staff of Texas Instruments components group; Texas Instruments; Fourth European edition; 1980; pp. 7/269-7/270 & 7/406-7/409.

4.34) Parity prediction in combinational circuits - B. Khodadad-Mostashiry; FTCS-9*; pp. 185-188.

4.35) as per 4.6); pp. 129-133 & 144-146.

4.36) as per 4.4); pp. 294-299.

4.37) as per 4.6); pp. 22-24.

4.38) b-adjacent error correction - D. C. Bossen; IBM J. Res. Devel.; Vol. 14 No. 4; July 1970; pp. 402-408.

4.39) Design for self-verification : An approach for dealing with testability problems in VLSI-based designs - R. M. Sedmak; 1979 Test Conf.*; pp. 112-120.

4.40) Redundancy by coding versus redundancy by replication for failure-tolerant sequential circuits - R. W. Larsen, I. S. Reed; IEEE Trans. Comput.; Vol. C-21, No.2; February 1972; pp. 130-137.

4.41) as per 4.6); pp. 46-48, 114-5, 118-9 & 133-140.

4.42) as per 4.6); pp. 133-140.

4.43) as per 4.6); pp. 37-46, 71-74, 114-115 & 123-124.

4.44) Arithmetic codes: Cost and effectiveness studies for application in digital system design - A. Avizienis; IEEE Trans. Comput.; Vol. C-20, No. 11; November, 1971; pp. 1322-1331.

4.45) Error codes for arithmetic operations - H. L. Garner; IEEE Trans. Electronic Comput.; Vol. EC-15, No. 5, May, 1966; pp. 763-770.

4.46) as per 4.6); pp. 40-46.

4.47) as per 4.6); pp. 38-46, 71-74, 114-115 & 147-148.

4.48) as per 4.6); pp. 48-50.

4.49) Modified Berger codes for the detection of unidirectional errors - H. Dong; FTCS-12*;pp. 317-320.

4.50) Design of fast self-testing checkers for a class of Berger codes - S. J. Piestrak;FTCS-15*;pp. 418-423.

4.51) On totally-self-checking checkers for separable codes - M. J. Ashjaee, S. M. Reddy; FTCS-7*; pp. 151-156.

4.52) Design of self-checking checkers for Berger codes - M. A. Marouf, A. D. Friedman;FTCS-8*;pp. 179-184.

4.53) as per 4.6); pp 24-34.

4.54) as per 4.7); pp. 256-265.

4.55) Memory finds and fixes errors to raise reliability of microcomputer - A. Heimlich, J. Korelitz; Electronics; January 3, 1980; pp. 168-172.

4.56) Application of shift register approach and its effective implementation - M. Kawai, S. Funastu, A. Yamada; 1980 Test Conf.*; pp. 22-25.

4.57) Enhancing testability of large-scale integrated circuits via test points and additional logic - M. J. Y. Williams, J. B. Angell; IEEE Trans. Comput.; Vol. C-22, No. 1; January, 1973; pp. 46-60.

4.58) Structured-test devices simplify test generation - S. Gupta; EDN; 14 November, 1985; pp.289-298.

4.59) A logic design structure for LSI testability - E. B. Eichelberger, T. W. Williams; 14th Design Automation Conference; New Orleans, Louisiana, USA; June 1977; IEEE Pub. No. 77 CH1216-1C; pp. 462-468.

4.60) Level-sensitive scan design test chips, boards, system - N. C. Berglund; Electronics; 15 March, 1979; pp. 108-110.

4.61) Design for testability of the IBM System/38 - L. A. Stolte, N. C. Berglund; 1979 Test Conf.*; pp. 29-36.

4.62) Design for testability - T. W. Williams, K. P. Parker; Proc. of the IEEE; Vol. 71, No. 1; January, 1983; pp. 98-112.

4.63) Testing logic networks and designing for testability - T. W. Williams, K. P. Parker; Computer; October, 1979; pp. 9-21.

4.64) Test generation techniques - S. B. Akers; Computer; March, 1980; pp. 9-15.

4.65) LSI logic testing - an overview - E. I. Muehldorf, A. D. Savkar; IEEE Trans. Comput.; Vol. C-30, No. 1; Janaury, 1981; pp. 1-16.

4.66) Incomplete scan path with an automatic test generation methodology - E. Trischler; 1980 Test Conf.*; pp. 153-162.

4.67) BIDCO, Built-in digital circuit observer - P. P. Fasang; 1980 Test Conf.*; pp. 261-265.

4.68) Built-in logic block observation techniques - B. Konemann, J. Mucha, G. Zwiehoff; 1979 Test Conf.*; pp. 37-317.

4.69) Built-in test for complex digital integrated circuits - B. Konemann, J. Mucha, G. Zwiehoff; IEEE J. of Solid-State circuits; Vol. SC-15, No. 3; June 1980; pp.315-318.

4.70) Design for autonomous test - E. J. McCluskey, S. Bozorgui-Nesbat; 1980 Test Conf.*; pp. 15-21.

4.71) Verification testing - A pseudoexhaustive test technique - E. J. McCluskey; IEEE Trans. Comput.; Vol. C-33, No. 6; June, 1984; pp. 541-545.

4.72) Structured design for testability to eliminate test pattern generation - S. Bozorgui-Nesbat, E. J. McCluskey; FTCS-10;* pp. 158-163.

4.73) Design for autonomous test - E. J. McCluskey, S. Bozorgui-Nesbat; IEEE Trans. Comput.; Vol. C-30, No. 11; November, 1981; pp. 866-874.

4.74) Autonomous testing and its application to testable design of logic circuits - H. Eiki, K. Inagaki, S. Yajima; FTCS-10*; pp. 173-178.

4.75) The TTL book for design engineers - The engineering staff of Texas Instruments components group; Texas Instruments; Fourth European edition; 1980; pp. 7:271-7:281.

4.76) Off-line, built-in test techniques for VLSI circuits - M. G. Buehler, M. W. Sievers; Computer; June, 1982; pp. 69-82.

4.77) Syndrome testable design of combinational circuits - J. Savir; IEEE Trans. Comput.; Vol. C-29, No. 6; June, 1980; pp. 442-451.

4.78) Radamacher Walsh spectral techniques: A new tool for digital network fault diagnosis - R. G. Bennetts, S. L. Hurst; FTCS-7; p. 213.

4.79) The logical processing of digital signals - S. L. Hurst; Crane, Russak, New York; 1978.

4.80) Testing by verifying Walsh coefficients - A. K. Susskind; FTCS-11*; pp.206-208.

4.81) Self-testing supercells: Alternative test strategies - D. K. Bhavsar;Autotestcon '80*;pp. 135-139.

4.82) Enhance computer fault isolation with a history memory - G. L. Fitzgerald; Autotestcon '80*; pp. 131-133.

4.83) as per 4.7); pp. 292-294.

4.84) Motorola MC6805P2 8-bit microcomputer unit; data sheet; Motorola Semiconductor Products Inc.; Scotland, UK; 1982

4.85) Let your next microcomputer check itself and cut down your testing overhead - J. Boney; Electronic Design; No. 18; 1 September 1979; pp. 100-105.

4.86) Motorola 6805 self test programs; Motorola Semiconductor Products Inc.; Scotland, UK.

4.87) Handling exceptions gracefully enhances software reliability - T. W. Starnes; Electronics; 11 September, 1980; pp. 153-157.

4.88) Motorola M68000; data sheet ADI-814-R1; Motorola Semiconductor Products Inc., USA; 1980.

4.89) Intel iAPX 43201/43202 VLSI general data processor; data sheet; Intel Corporation ;1981.

4.90) Error reporting in the Intel iAPX 432 - D. B. Johnson; FTCS-14*; p. 24-28.

4.91) Diagnostic PROMs enhance PCB testability - J. Gabris; New Electronics;11 December 1984;pp. 37-43.

4.92) Monolithic Memories 53DA841/63DA841 2048 x 4 diagnostic registered PROM; data sheet; Monolithic Memories.

4.93) Semicustom-logic suppliers differ on how to best deal with testability - W. Twaddell; EDN; 24 September, 1982; pp. 69-73.

4.94) 'Expandable' IC finds and fixes errors fast - C. Carinalli, M. Evans; Electronic Design; 18 February 1982; pp. 177-186.

4.95) Structured arrays mix gate arrays and standard cells - R. Rasmussen; New Electronics; 18 February 1986; pp. 40-42.

4.96) Gate array test circuits generated automatically - J. Vickerton; New Electronics; 18 February, 1986; pp. 58-60.

4.97) Implementation techniques for self-verification - R. M. Sedmak; 1980 Test Conf.*; pp. 267-278.

4.98) Fail-safe computers increase reliability, lower costs - G. Kravetz; Mini-Micro Systems; April, 1984; pp. 121-130.

* see section B.5.

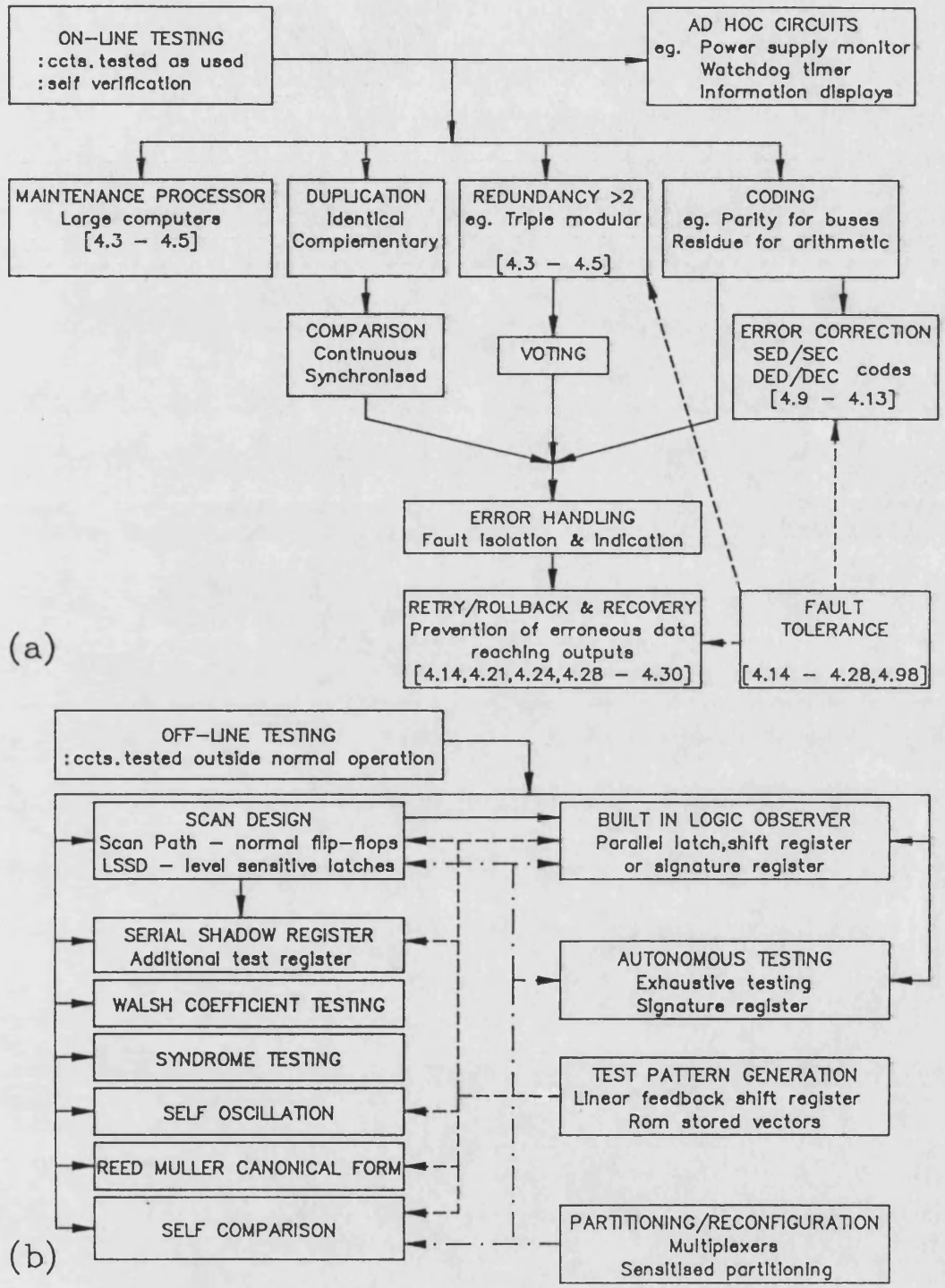
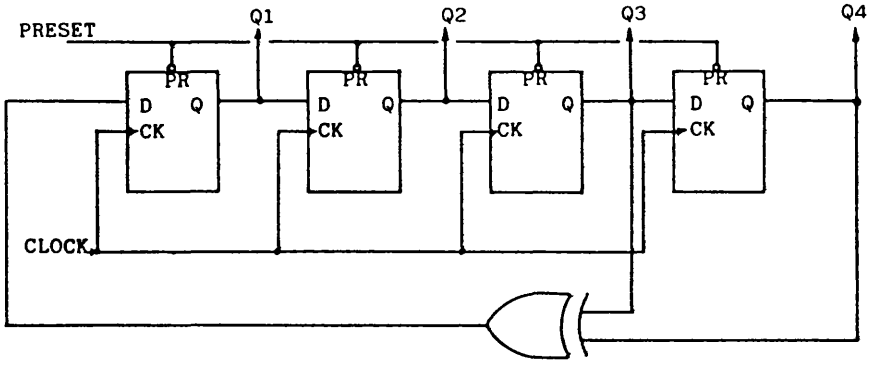


FIGURE 4.1 ON AND OFF LINE TECHNIQUES FOR BUILT IN TEST



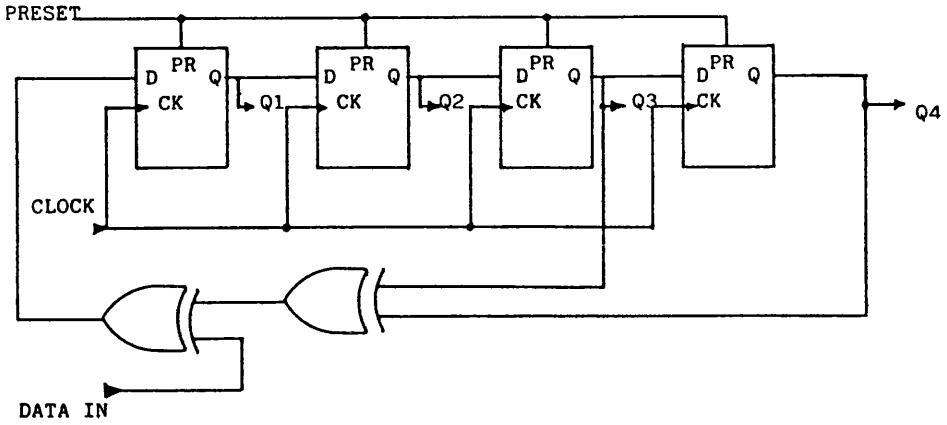
(a)

N= 4 so number of patterns = 15

SEQUENCE	Q1	Q2	Q3	Q4	
1	1	1	1	1	← Preset
2	0	1	1	1	
3	0	0	1	1	
4	0	0	0	1	
5	1	0	0	0	
6	0	1	0	0	
7	0	0	1	0	
8	1	0	0	1	
9	1	1	0	0	
10	0	1	1	0	
11	1	0	1	1	
12	0	1	0	1	
13	1	0	1	0	
14	1	1	0	1	
15	1	1	1	0	
1	1	1	1	1	

(b)

FIGURE 4.2 **THE LINEAR FEEDBACK SHIFT REGISTER**



(a)

DATA IN	Q1	Q2	Q3	Q4	
1	1	1	1	1	← Preset
0	1	1	1	1	
1	0	1	1	1	
0	1	0	1	1	
1	0	1	0	1	
1	0	0	1	0	
1	0	0	0	1	
0	0	0	0	0	
0	0	0	0	0	
1	0	0	0	0	
0	1	0	0	0	
1	0	1	0	0	
1	1	0	1	0	
0	0	1	0	1	
0	1	0	1	0	
0	1	1	0	1	
1	1	1	1	0	
	0	1	1	1	← Final signature

(b)

FIGURE 4.3 LFSR DATA COMPRESSOR (SIGNATURE REGISTER)

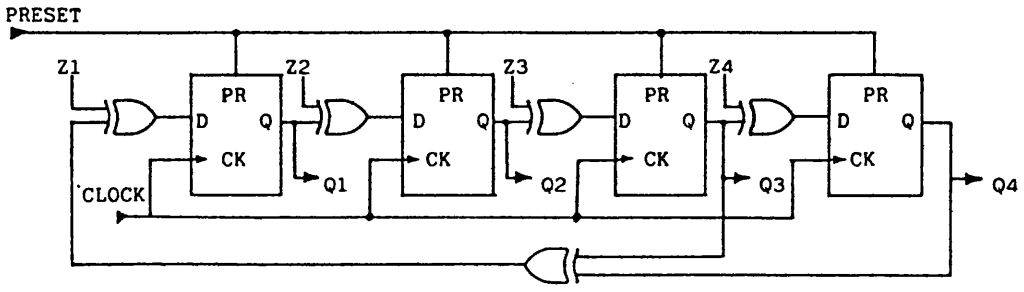


FIGURE 4.4 MULTI-INPUT SIGNATURE REGISTER (MISR)

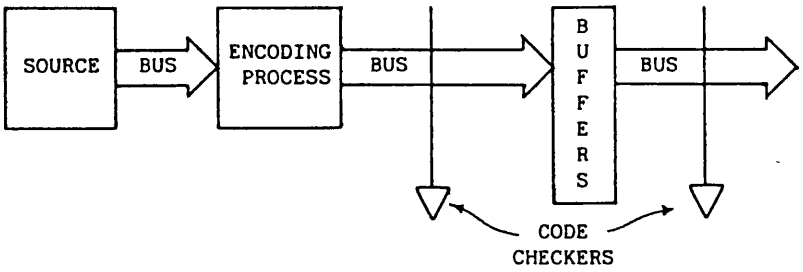
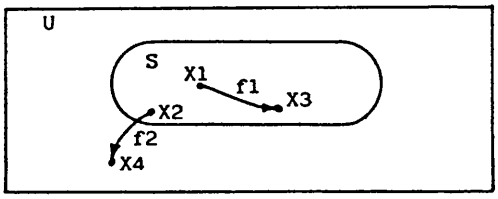


FIGURE 4.5 CODE GENERATION & CHECKING



X1, X2 and X3 are codewords : X4 is a non codeword

f1 is an undetectable error : f2 is a detectable error

FIGURE 4.6 FAILURES IN AN ERROR DETECTING CODE

FIGURE 4.7 GENERATION OF ODD AND EVEN PARITY

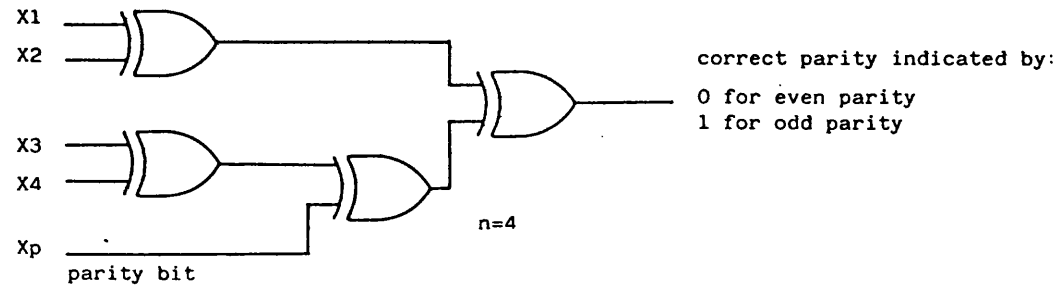


FIGURE 4.8 CHECKING SCHEME FOR PARITY - EOR TREE

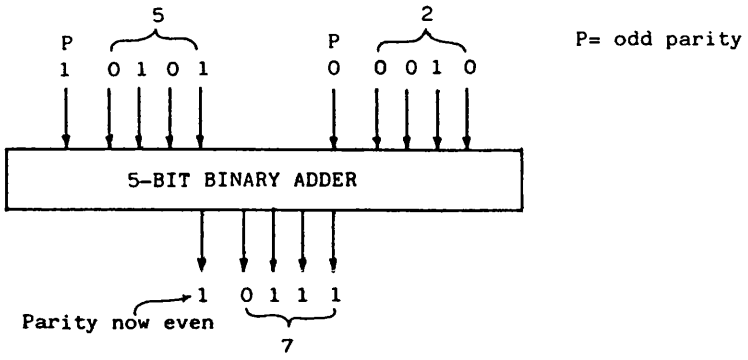


FIGURE 4.9 PARITY DURING AN ARITHMETIC PROCESS

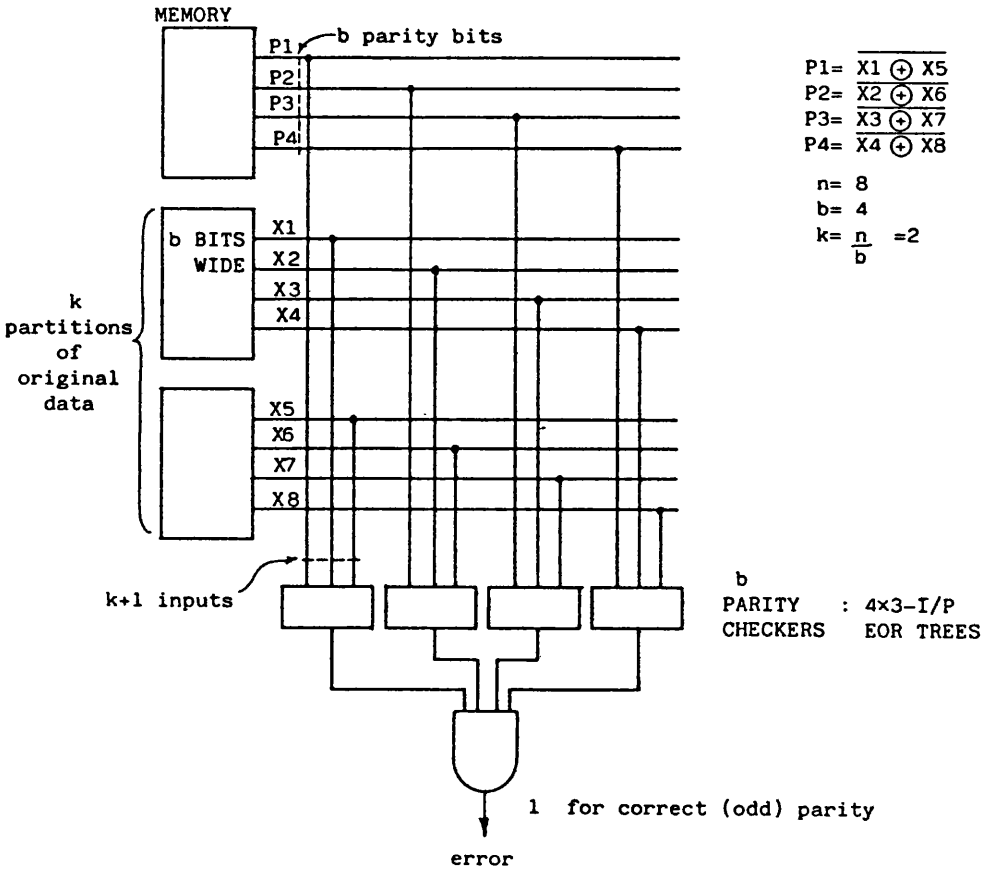


FIGURE 4.10 CHECKING SCHEME FOR B-ADJACENT CODES

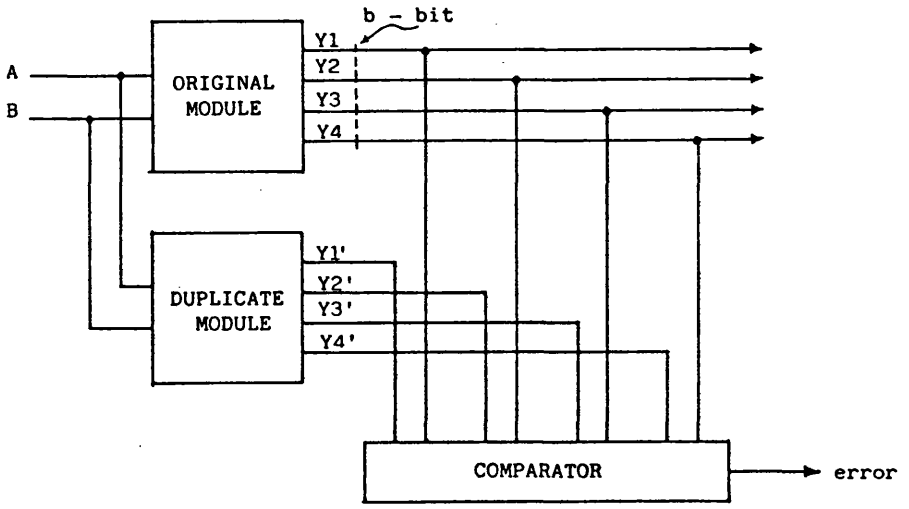


FIGURE 4.11 DUPLICATION AND COMPARISON

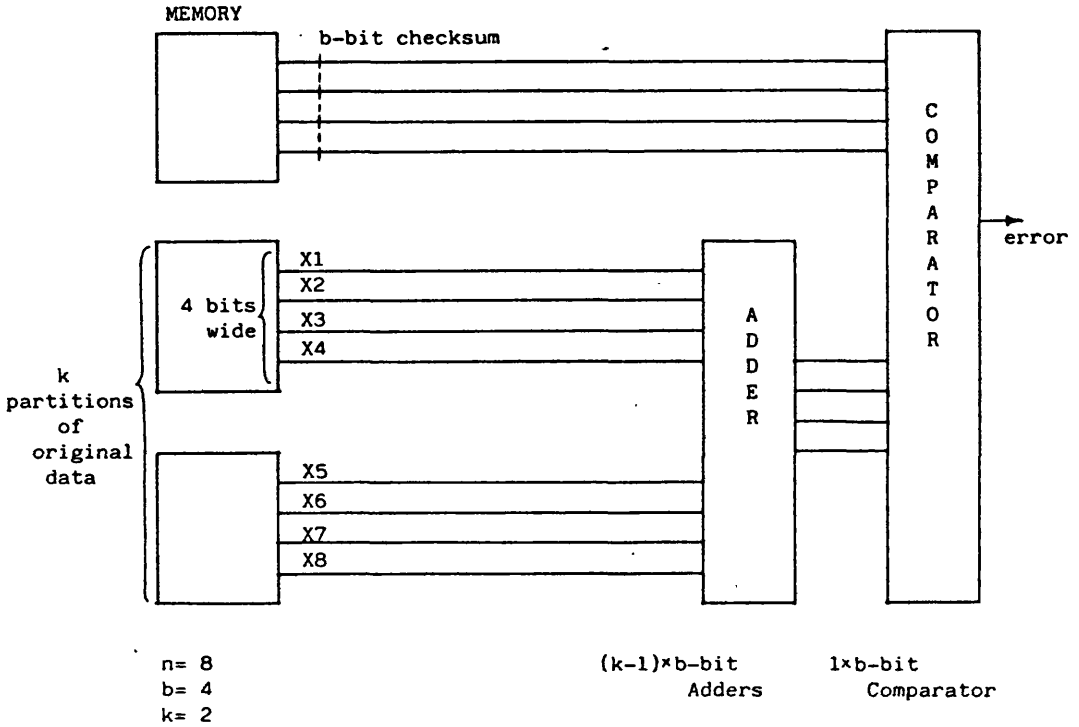


FIGURE 4.12 CHECKING SCHEME FOR CHECKSUM CODES-HARDWARE

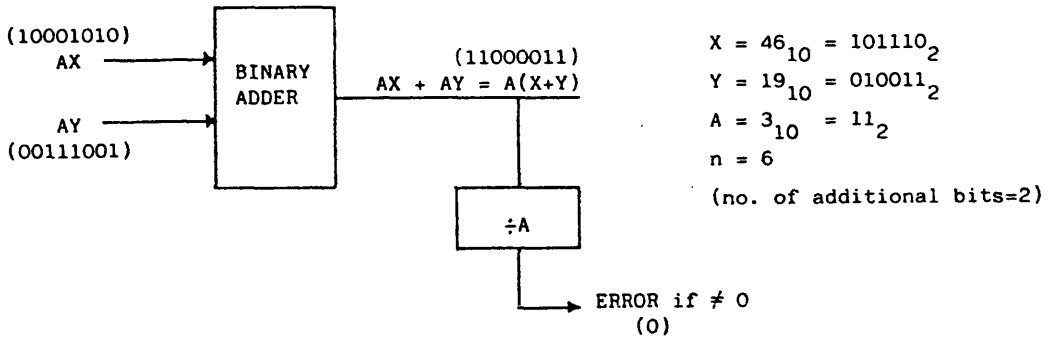


FIGURE 4.13 CHECKING SCHEME FOR AN CODES

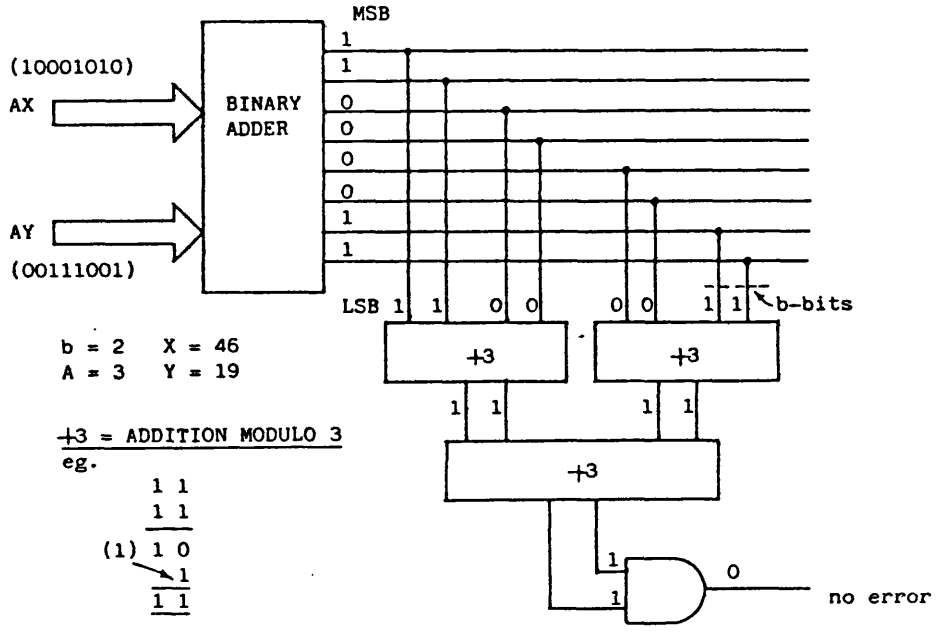


FIGURE 4.14 CHECKING SCHEME FOR LOW COST AN CODES

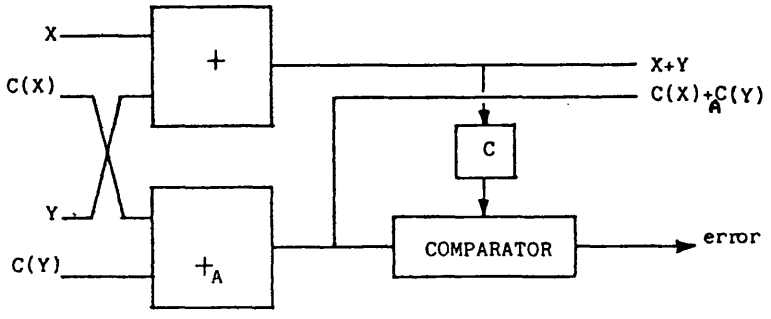


FIGURE 4.15 CHECKING SCHEME FOR RESIDUE CODES

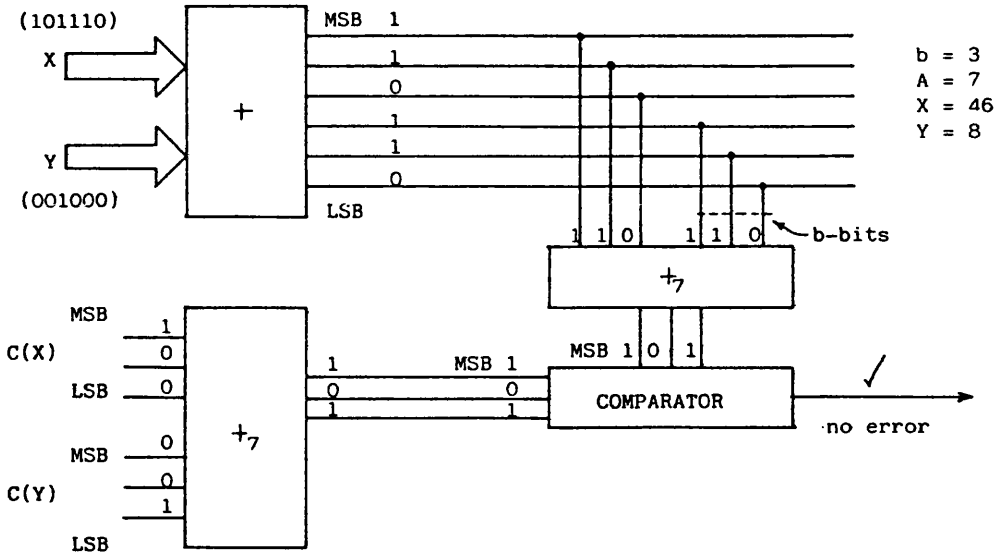


FIGURE 4.16 CHECKING SCHEME FOR LOW COST RESIDUE CODES

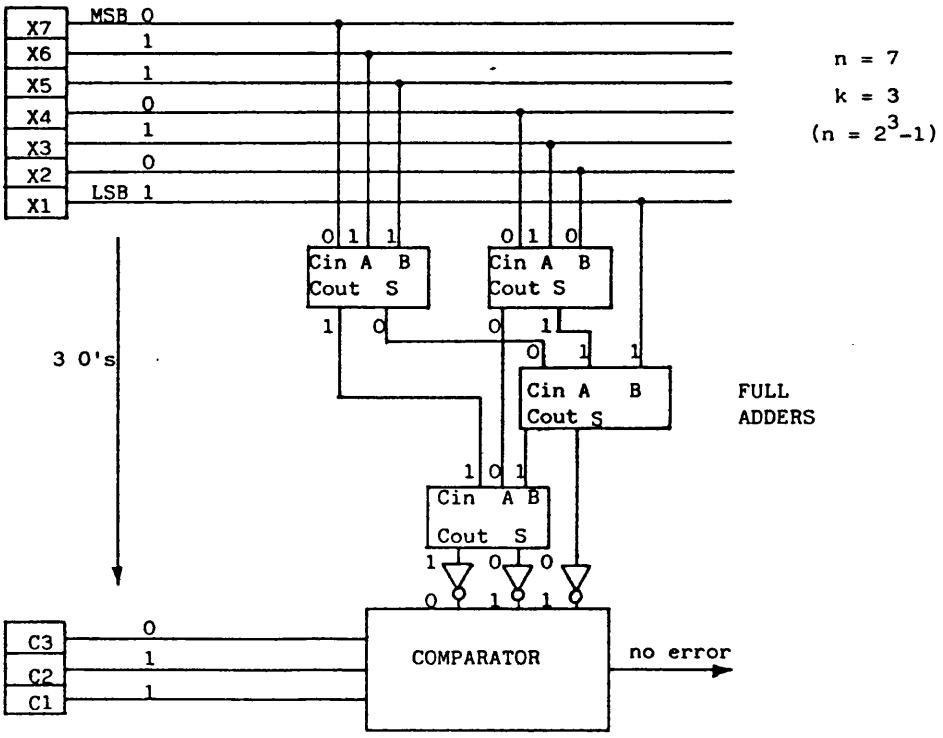


FIGURE 4.17 CHECKING SCHEME FOR MAXIMAL LENGTH BERGER CODES

INFORMATION BITS					CODEWORD						
M2	M1	MO	M(x)	V(x)=M(x)G(x)	V6	V5	V4	V3	V2	V1	VO
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	$x^4+x^3+x^2+1$	0	0	1	1	1	0	1
0	1	0	x	$x^5+x^4+x^3+x$	0	1	1	1	0	1	0
0	1	1	x+1	x^5+x^2+x+1	0	1	0	0	1	1	1
1	0	0	x^2	$x^6+x^5+x^4+x^2$	1	1	1	0	1	0	0
1	0	1	x^2+1	$x^6+x^5+x^3+1$	1	1	0	1	0	0	1
1	1	0	x^2+x	$x^6+x^3+x^2+x$	1	0	0	1	1	1	0
1	1	1	x^2+x+1	x^6+x^4+x+1	1	0	1	0	0	1	1

$G(x) = x^4+x^3+x^2+1$ n = 3 k = 7 r = 4

FIGURE 4.18 GENERATION OF A (7,3) NON-SYSTEMATIC CYCLIC CODE

INFORMATION BITS					CODEWORD $V(x)=x^4M(x)-C(x)$						
M2	M1	MO	$x^4M(x)$	$C(x)=REM(x^4M(x)/G(x))$	V6	V5	V4	V3	V2	V1	VO
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	x^4	x^3+x^2+1	0	0	1	1	1	0	1
0	1	0	x^5	x^2+x+1	0	1	0	0	1	1	1
0	1	1	x^5+x^4	x^3+x	0	1	1	1	0	1	0
1	0	0	x^6	x^3+x^2+x	1	0	0	1	1	1	0
1	0	1	x^6+x^4	x+1	1	0	1	0	0	1	1
1	1	0	x^6+x^5	x^3+1	1	1	0	1	0	0	1
1	1	1	$x^6+x^5+x^4$	x^2	1	1	1	0	1	0	0

$G(x)=x^4+x^3+x^2+1$

original information bits

FIGURE 4.19 GENERATION OF A (7,3) SYSTEMATIC CYCLIC CODE

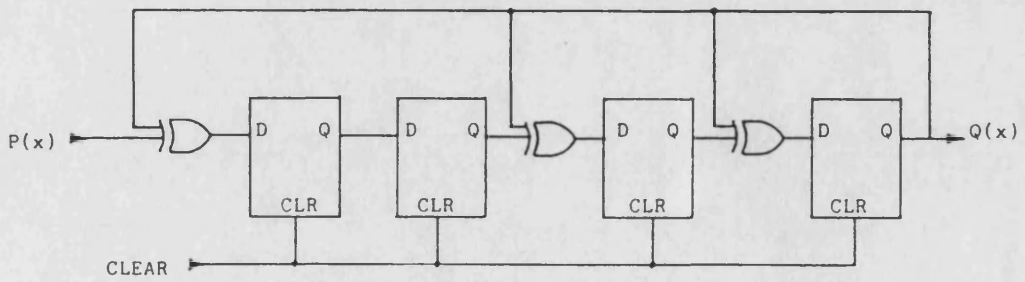


FIGURE 4.20 A LFSR FOR DIVIDING BY $x^4+x^3+x^2+1$

d	e	f
1	0	0
2	1	0
3	2	1
•	•	•
d	d-1	$\left\lceil \frac{d-1}{2} \right\rceil$

d= Hamming distance
e= no. of errors detected
f= no. of errors corrected

FIGURE 4.21 CAPABILITIES OF A CODE WITH HAMMING DISTANCE d

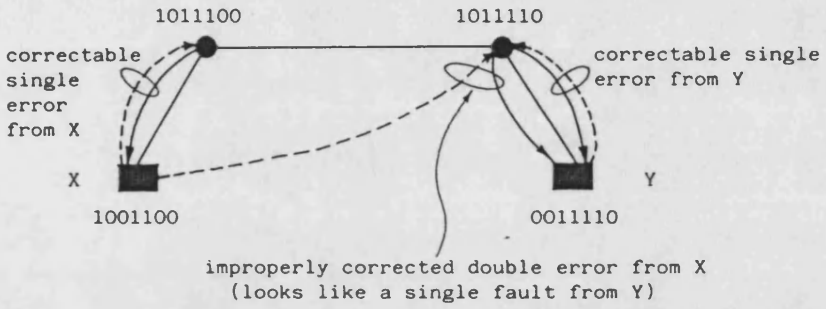


FIGURE 4.22 FRAGMENT OF A DISTANCE 3 ERROR CORRECTING CODE

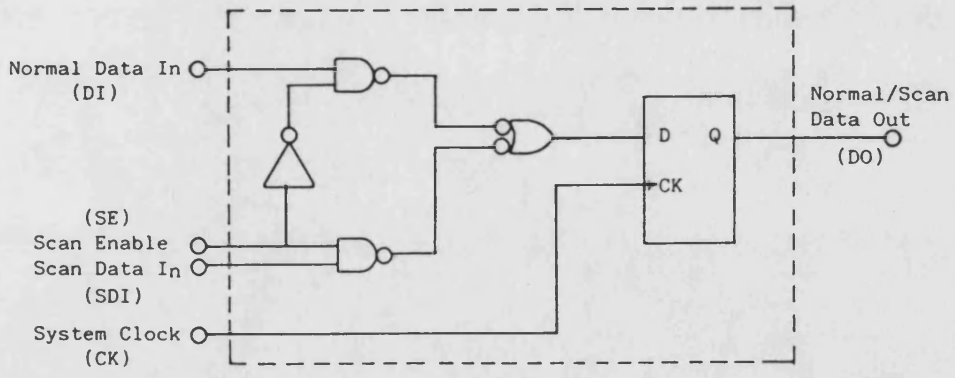


FIGURE 4.23 D-TYPE FLIP FLOP FOR SCAN PATH DESIGN

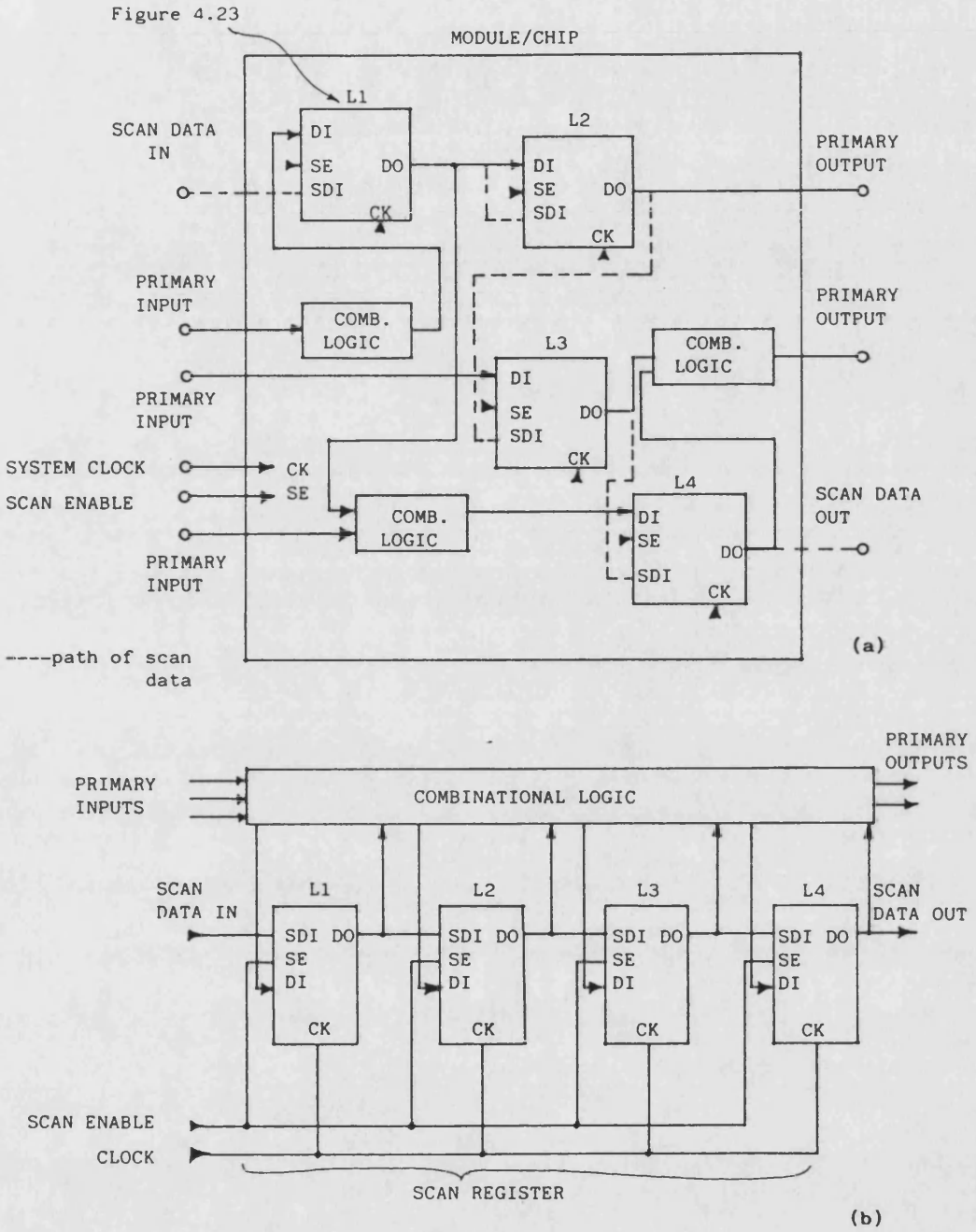


FIGURE 4.24 TYPICAL SCAN PATH STRUCTURE

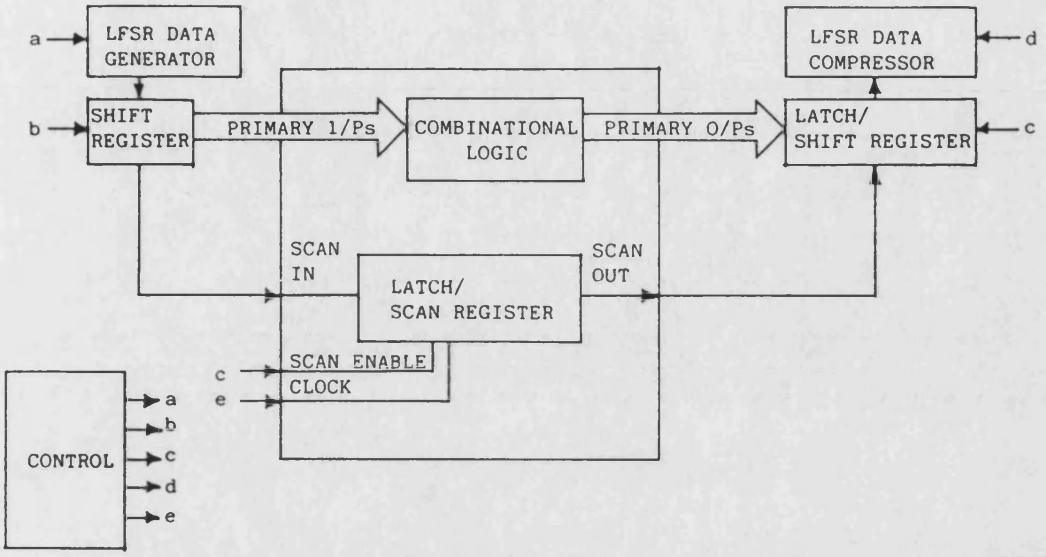


FIGURE 4.25 SCAN DESIGN WITH LFSR DATA GENERATION & COMPRESSION

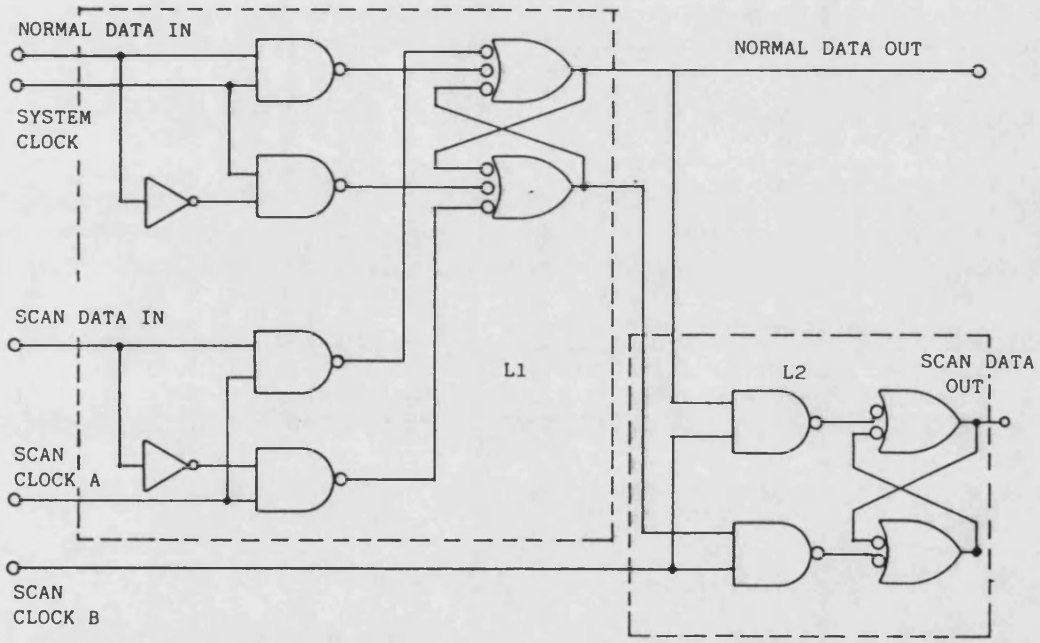


FIGURE 4.26 SHIFT REGISTER LATCH (SRL) FOR LSSD

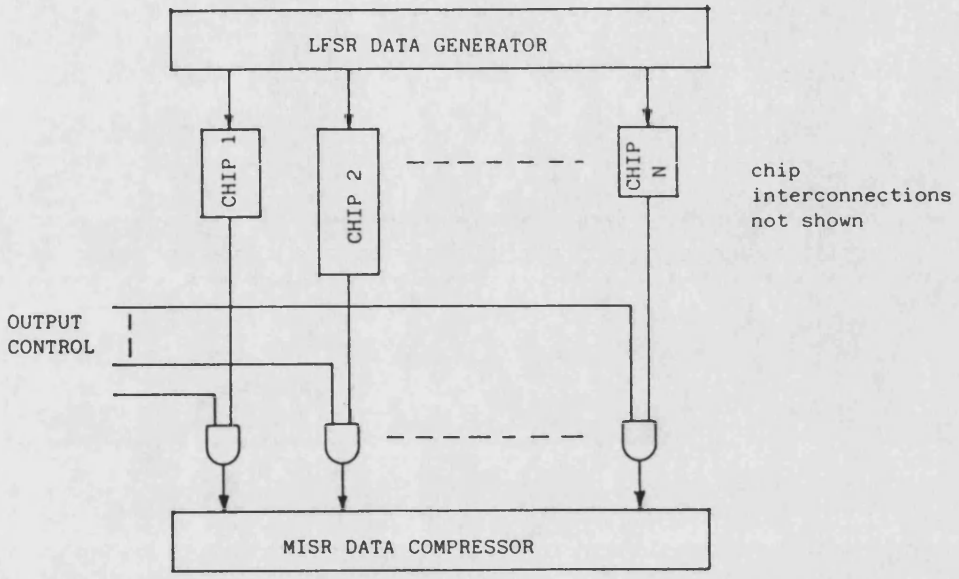
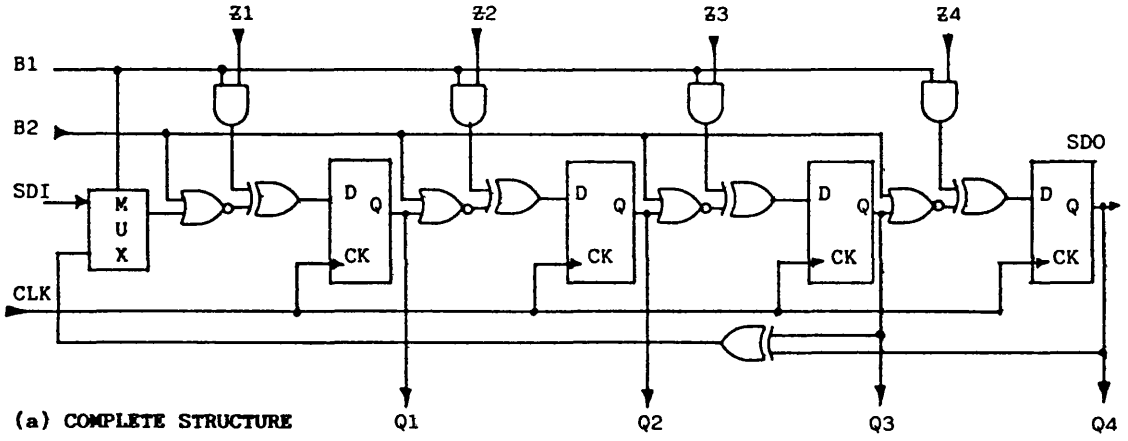
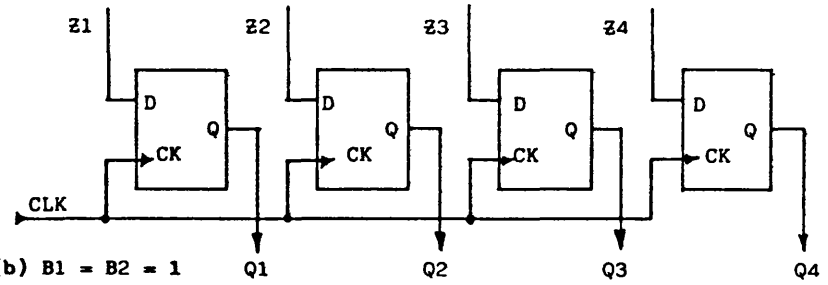


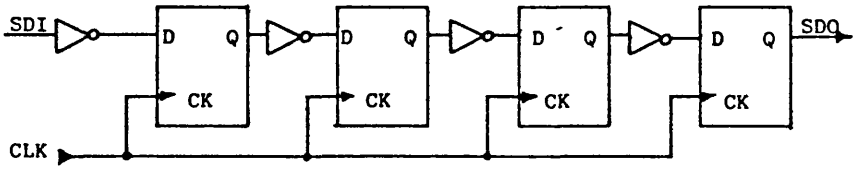
FIGURE 4.27 STUMPS



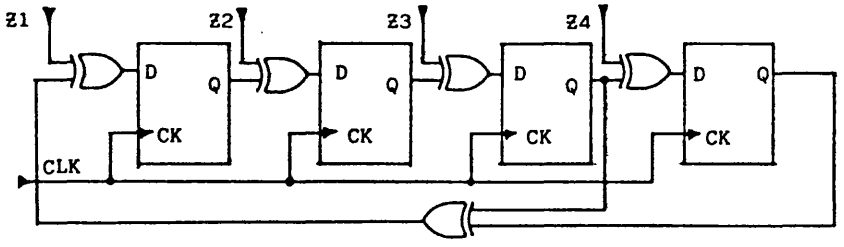
(a) COMPLETE STRUCTURE



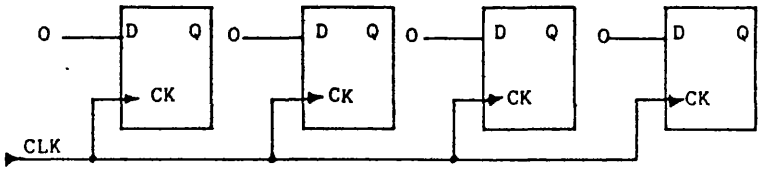
(b) B1 = B2 = 1



(c) B1 = B2 = 0



(d) B1 = 1, B2 = 0



(e) B1 = 0, B2 = 1

FIGURE 4.28 BUILT IN LOGIC BLOCK OBSERVER

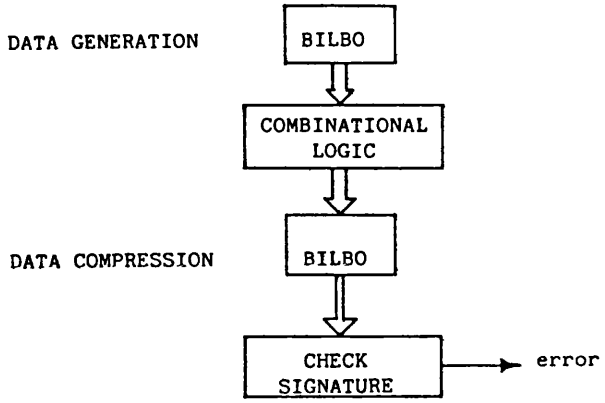


FIGURE 4.29 TESTING WITH BILBOS

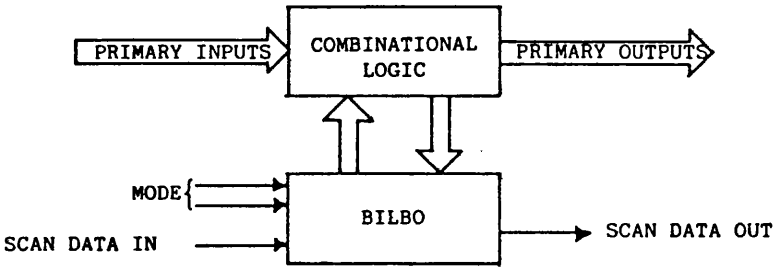


FIGURE 4.30 SCAN DESIGN WITH A BILBO

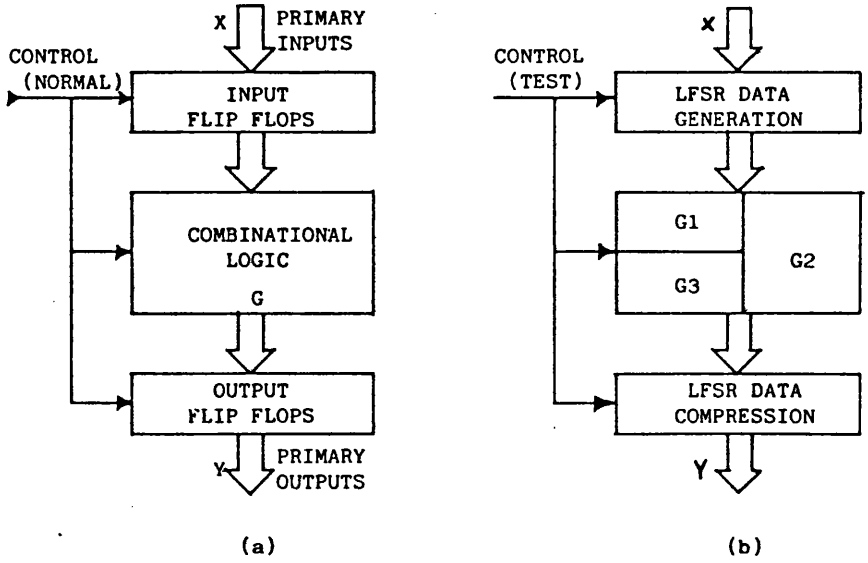
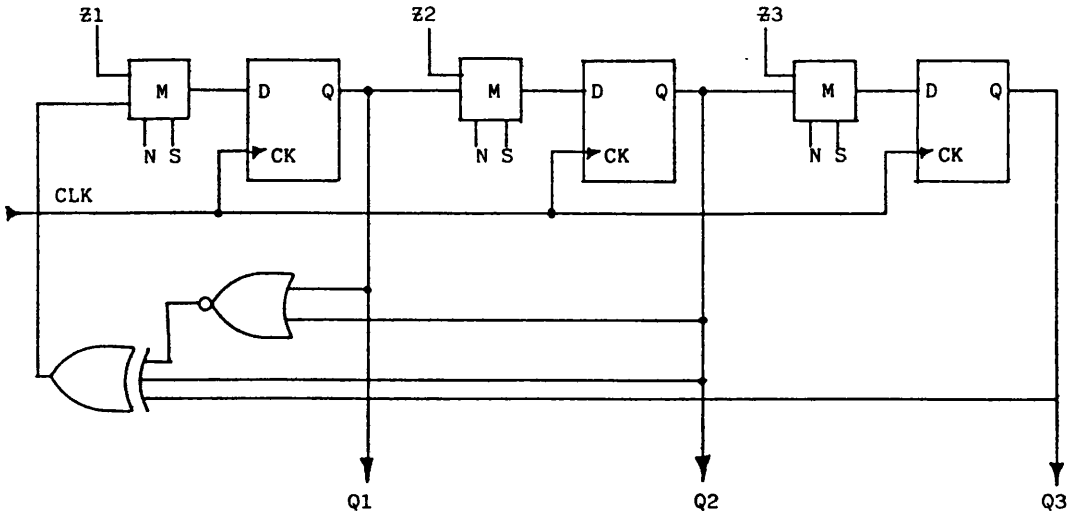


FIGURE 4.31 AUTONOMOUS TESTING



N	S	MODE - M
1	x	NORMAL OPERATION
0	0	LFSR DATA GENERATOR
0	1	LFSR DATA COMPRESSOR

FIGURE 4.32 **LFSR FOR AUTONOMOUS TESTING**

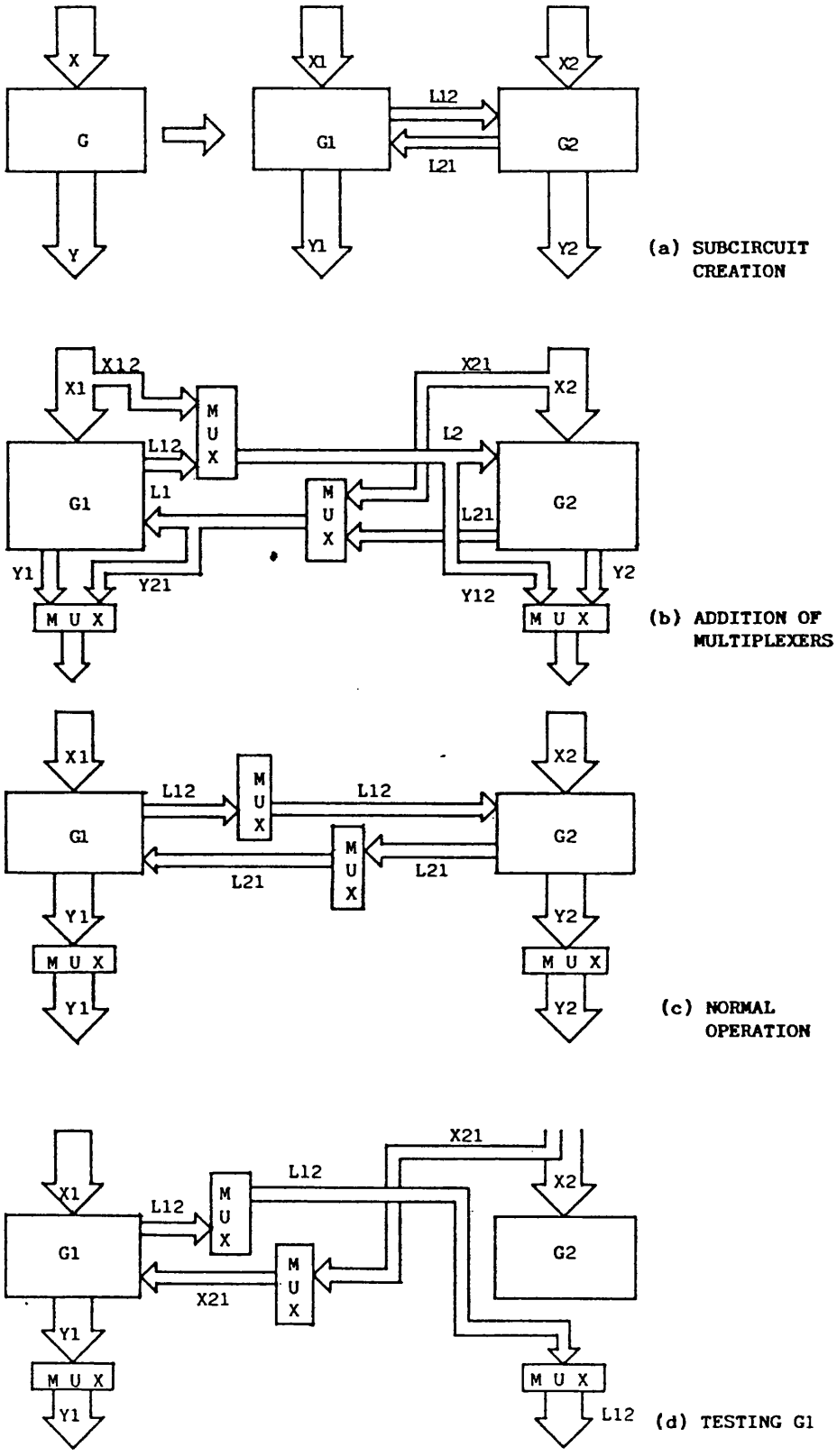


FIGURE 4.33 PARTITIONING WITH MULTIPLEXERS

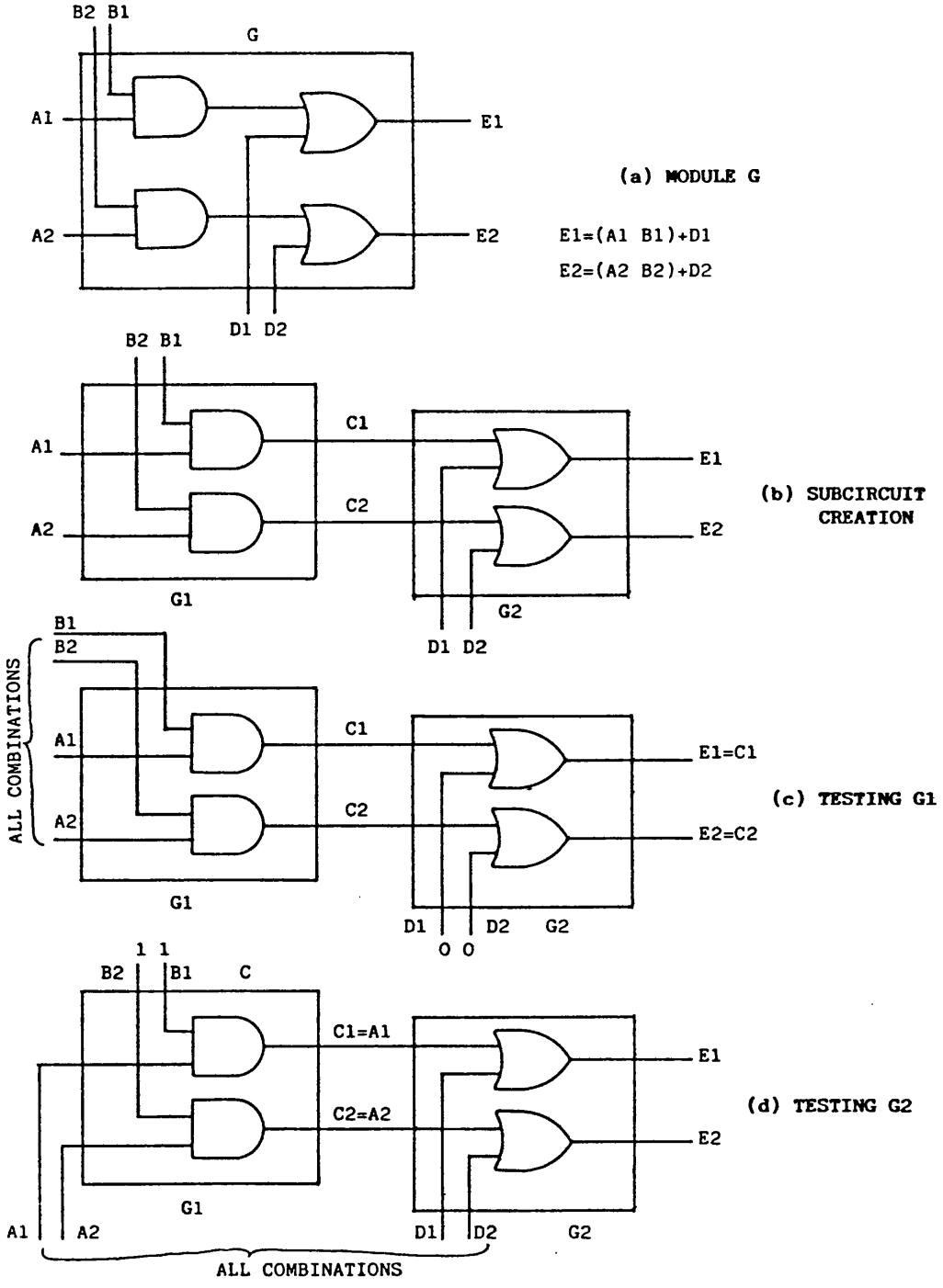


FIGURE 4.34 SENSITISED PARTITIONING

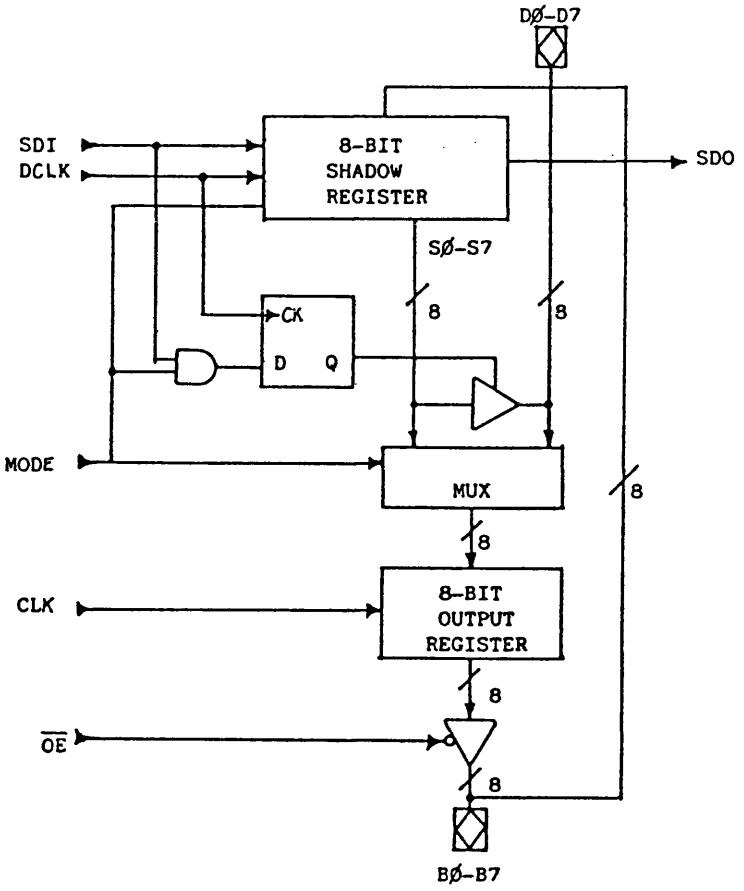


FIGURE 4.35 STRUCTURE OF THE SERIAL SHADOW REGISTER

INPUTS				OUTPUTS			OPERATION
MODE	SDI	CLK	DCLK	B0-B7	S0-S7	SDO	
0	X	↑	^	Dn→Bn	HOLD	S7	LOAD OUTPUT REGISTER FROM INPUT BUS
0	X	^	↑	HOLD	Sn-1→Sn SDI→S0	S7	SHIFT SHADOW REGISTER DATA
0	X	↑	↑	Dn→Bn	Sn-1→Sn SDI→S0	S7	LOAD OUTPUT REGISTER FROM INPUT BUS WHILST SHIFTING SHADOW REGISTER DATA
1	X	↑	^	Sn→Bn	HOLD	SDI	LOAD OUTPUT REGISTER FROM SHADOW REGISTER
1	0	^	↑	HOLD	Bn→Sn	SDI	LOAD SHADOW REGISTER FROM OUTPUT BUS
1	0	↑	↑	Sn→Bn	Bn→Sn	SDI	SWAP SHADOW REGISTER AND OUTPUT REGISTER
1	1	^	↑	HOLD	HOLD	SDI	NO OPERATION : D0-D7 ARE OUTPUTS

↑ = rising edge of clock ^ = clock steady or falling

FIGURE 4.36 FUNCTIONS OF THE SERIAL SHADOW REGISTER

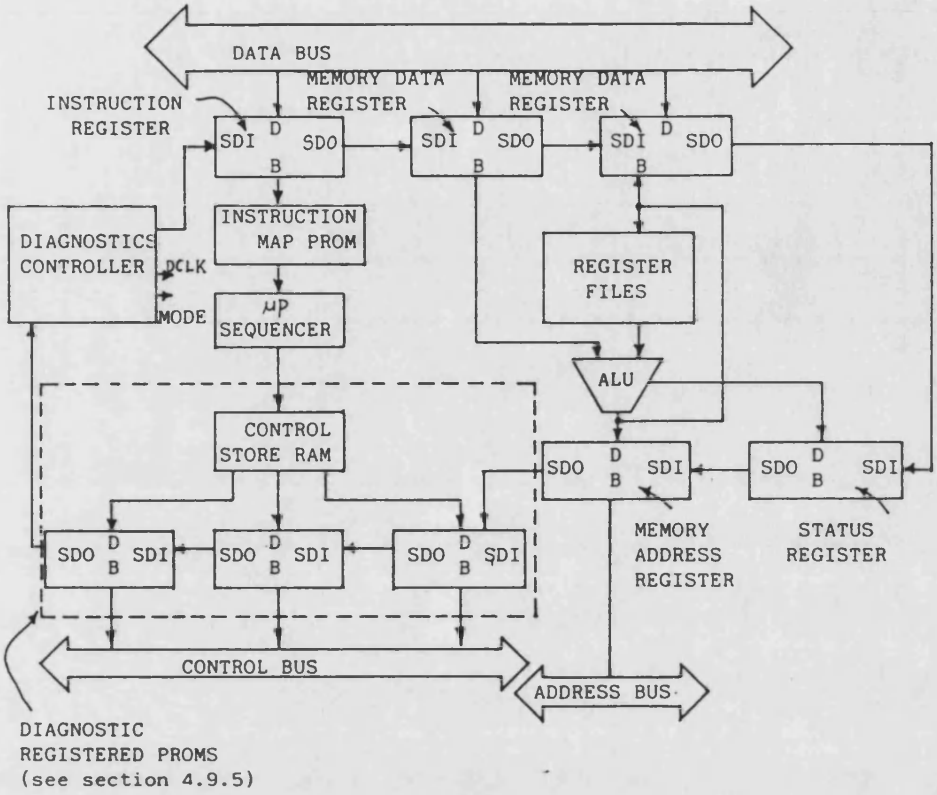


FIGURE 4.37 GENERAL PURPOSE CPU WITH SSR

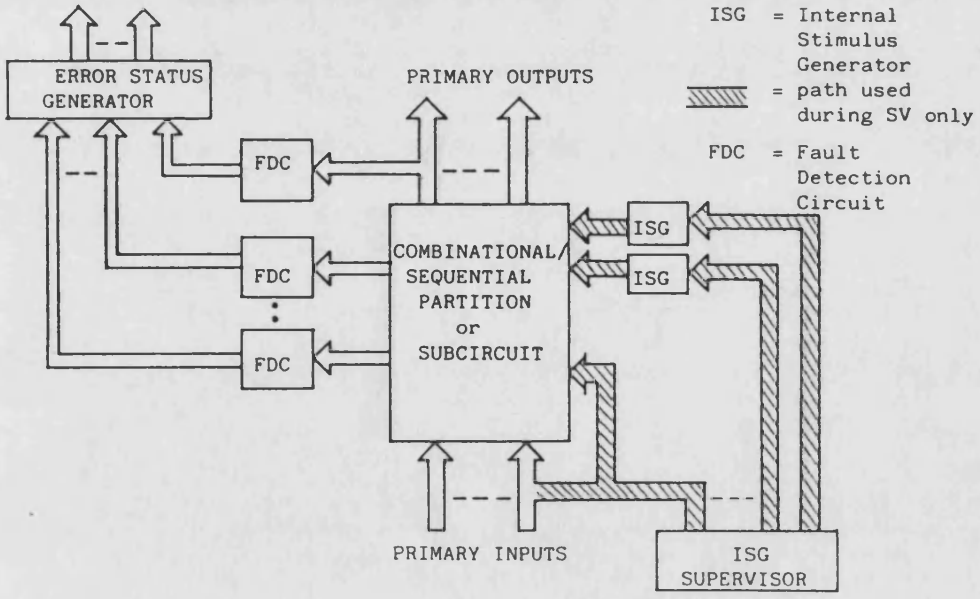


FIGURE 4.38 SELF VERIFICATION

CHAPTER FIVE : SELF CHECKING CIRCUITS - AN INTRODUCTION

5.1 : INTRODUCTION

A circuit which has its output encoded in an error detecting code is termed a self checking circuit [5.1, 5.2], as shown in Fig 5.1. During failure free operation its output is always a code word (see section 4.3). The output of a self checking circuit is monitored by a checker which indicates the presence of non codewords, as shown in Fig. 5.2. A self checking circuit has the properties of 'self test' and 'fault security'. These are defined by Anderson [5.2] as follows.

DEFINITION 5.1 : A circuit is self testing if, for every fault from a defined set, it produces a non codeword output for at least one codeword input.

DEFINITION 5.2 : A circuit is fault secure if, for every fault from a defined set, it never produces an incorrect codeword output for codeword inputs.

DEFINITION 5.3 : A circuit is totally self checking if it is both self testing and fault secure.

A circuit could be self testing and fault secure for different input sets and different fault sets, so the following is proposed. A circuit is self testing for an input set of codewords X and fault secure for an input set of codewords N, where N is a subset of X. If N equals X, then the circuit is totally self checking as defined above. If N is a null set then the circuit is self testing only and not fault secure at all. If N is a non null subset of X then the circuit is partially self checking.

The self testing and fault secure properties of sequential circuits can be similarly defined as follows.

DEFINITION 5.4 : A sequential circuit is self testing if, for every fault from a defined set, it produces a non codeword output for at least one codeword input.

DEFINITION 5.5 : A sequential circuit is fault secure if, for every fault from a defined set, it never produces an incorrect codeword output for codeword inputs.

Only combinational circuits are considered during the first part of the Chapter, with formal definitions of totally self checking, partially self checking and self testing only circuits. Some examples of totally self checking circuits are described in section 5.3. In addition, sufficient conditions are given for a network to be totally self checking. Totally self checking checkers are essential for checking codes, so section 5.4 details their general characteristics and design requirements, together with checkers specifically for separable codes. Section 5.5 presents the partially self checking property and its use for networks performing logical operations. Section 5.6 discusses self testing only circuits with sufficient conditions for self testing only networks. Section 5.7 analyses the self testing and fault secure properties for bit and byte-sliced circuits, whilst section 5.8 describes self checking sequential circuits.

Parallel processing of information (encoding and decoding) is assumed throughout, since this is the most appropriate method for computers.

5.2 : DEFINITION OF TERMS

Combinational circuits are considered to produce a fault free output vector $Z(X_i)$, where X_i is an input vector, see Fig. 5.3a. With a fault f in the circuit this becomes $Z^f(X_i)$, as shown in Fig. 5.3b. Faults will be one or more lines stuck at a logic value of 1 or 0 (SA1 or SA0); i.e. single and multiple faults. Unidirectional multiple faults have all affected lines stuck at the same value.

A circuit whose output $Z(X_i)$ is encoded in an error detecting code S is referred to as a self checking circuit [5.1,5.2], see Fig. 5.3c. During fault free operation, the output of such a circuit is always a codeword S_i, where:

$$S_i = Z(X_i) \text{ and } S_i \in S \qquad (5.1)$$

The set S is termed the output code space or output code set. A checker monitors the output of the circuit , detecting and indicating the presence of non codewords. A fault can change the output into a non codeword (detectable), or an incorrect codeword (undetectable). The fault secure property of self checking circuits limits the occurrence of undetectable errors.

DEFINITION 5.6 : A circuit is fault secure for an input set N and a fault set F_s if, $\forall f \in F_s$ and $\forall N_i \in N$, then $Z^f(N_i) = Z(N_i) \in S$ or $Z^f(N_i) \notin S$.

Thus the output of a fault secure circuit is always correct, or is a non codeword, as indicated in Fig. 5.4. The input set N is termed the secure input set (not necessarily all code inputs) and F_s the secure fault set. Fault security ensures that a fault will not cause an undetectable error, but does not indicate that it will eventually produce a detectable error; this is covered by the self testing property for self checking circuits.

DEFINITION 5.7 : A circuit is self testing for an input set X and a fault set F_t if, $\forall f \in F_t$, $\exists X_i \in X$ such that $Z^f(X_i) \notin S$.

Fig. 5.5 illustrates this definition. The input X_i for which $Z^f(X_i)$ is not in S is termed a test for f and F_t is the tested fault set.

X is termed the normal input set and it is assumed that

all X_i occur during normal operation. Then the self testing property guarantees that all faults in F_t will produce detectable errors.

U is the set of all possible n-bit vectors U_i, given by:

$$U = \{ U_i = \langle u_1 u_2 \ldots u_n \rangle \mid u_j \in \{0,1\}, 1 \leqslant j \leqslant n, 1 \leqslant i \leqslant 2^n \} \qquad (5.2)$$

which is abbreviated to:

$$U = \{0,1\}^n : \text{the set of all n-bit combinations of 1 and 0} \qquad (5.3)$$

Thus:

$$X \subseteq U \quad \text{and} \quad S \subset U \qquad (5.4)$$

The secure input set N is assumed to be a subset of X, $N \subseteq X$. Fault security specifies circuit behaviour for codeword inputs only. Although a circuit may be fault secure for inputs outside N, these are not of interest since they do not occur during normal operation. F_s is also assumed to be a subset of F_t, $F_s \subseteq F_t$.

The properties of self testing and fault security are also illustrated in Fig. 5.6. This depicts the set of all faults F with its subsets F_s and F_t, the set of all input vectors U with subsets X and N and the set of all output vectors with subset S. Self test is demonstrated by the existence of a test in X for the faults f1, f2 and f3 in F_t (X_1 or X_2, X_3 and X_1 respectively). Fault security is illustrated by the behaviour of various faults on $Z(X_i)$. Erroneous codewords may be produced from faults outside F_s (eg. $Z^{f3}(X_2)$) or inputs outside N (eg. $Z^{f2}(X_1)$).

Fault security specifies the behaviour of a circuit for codeword inputs only. It is also important to know how circuits perform with non codeword inputs. The code disjoint property covers this aspect of operation.

DEFINITION 5.8 : A circuit is code disjoint for an input code S' and an output code S if, $\forall X_i \notin S'$ then $Z(X_i) \notin S$.

The fault free output function of a circuit which is code disjoint maps codeword inputs $X_i \in S'$ to codeword outputs $S_i \in S$ and maps non codeword inputs $X_i \notin S'$ to non codeword outputs $S_i \notin S$. Fig. 5.7 illustrates this property.

Given X, F_t is determined to be the largest fault set for which the circuit is self testing. F_s is chosen on the basis of faults that are likely to occur (eg. all single stuck-at faults) and then N is determined to be the largest input set for which the circuit is fault secure. N is therefore controlled by the choice of F_s, but depending on whether N contains all, some or none of X, then the circuit is defined as totally self checking, partially self checking, or self testing only. These are defined as follows.

DEFINITION 5.9 : A totally self checking (TSC) circuit is self testing for fault set F_t with normal input set X and fault secure for fault set F_s with input set X.

In a TSC circuit, all faults in F_t are tested and no fault in F_s causes an undetectable error during normal operation However, there are circuits which are fault secure for only a subset of normal inputs.

DEFINITION 5.10 : A partially self checking (PSC) circuit is self testing for fault set F_t with normal input set X, but fault secure for fault set F_s with a non null subset N of X.

In a PSC circuit, all faults in F_t are testing during normal operation. Faults in F_s will not cause undetectable errors when inputs are from N, but can when inputs are from the set X-N.

Finally, a particular choice of F_s might result in there being no inputs for which the circuit is fault secure.

DEFINITION 5.11 : A self testing only (STO) circuit is self testing for fault set F_t with normal input set X, but fault secure for fault set F_s with a null subset N of X.

In a STO circuit, all faults in F_t are tested during normal operation, but undetectable errors can occur at any time, as shown in Fig. 5.8.

The operation of these three forms of a self checking circuit can be summarised as follows:

1) TSC circuits and PSC circuits operating in their secure mode (inputs from N) always output correct codewords for any fault in F_s.
2) STO circuits and PSC circuits operating in their insecure mode (inputs from X-N) can output incorrect codewords.
3) All self checking circuits have the self testing property, which guarantees that any fault from F_t will eventually be detected (eventually since the occurrence of input X_i from X, which produces a detectable error for fault f, cannot be specified).

The likelihood of an undetectable error during the insecure mode of a PSC circuit is dependent on its usage in that mode. If the insecure mode is used infrequently, it is likely that a fault will be detected during its secure mode before any erroneous result is produced during its insecure mode. Under these conditions, even undetectable errors occurring during its insecure mode will soon be detected when it reverts to the secure mode.

There is always a chance that transient errors occurring during an insecure mode will never be detected, so, for critical applications, the use of PSC or STO circuits may

be ruled out. In non critical applications an immediate detection of errors is not essential, only an eventual detection is required. Here low cost STO and PSC circuits can be employed instead of the more expensive TSC circuits.

Another point worth considering is the probability that a fault will be detected within t units of time from its occurrence. The expected value of t has been referred to as the 'error latency' of a circuit [5.3]. Hardware detection of single failures is only vaild if the error latency of the circuit is much less than the expected time for a second fault to occur. The circuit might otherwise experience a multiple fault for which it is insecure.

5.3 : TOTALLY SELF CHECKING CIRCUITS AND NETWORKS

In totally self checking (TSC) circuits, the set of secure inputs N is the same as the normal input set X, so only the latter needs to be mentioned. A number of simple TSC circuits are presented, which either process their inputs without modification, or transform them into other code-words. These and other TSC circuits may be interconnected to form a network, but this network will, in general, not be TSC. Sufficient requirements are therefore given to ensure that it is.

5.3.1 : TSC Circuits

Example 5.1 : The k-bit parity buffer shown in Fig. 5.9, which consists of k identical buffer gates, is TSC. The output code space S equals the normal input set X and is the set of all odd parity k-bit vectors. A single stuck-at fault will either have no effect on the output, or change it from an odd parity codeword to a non codeword, since a single bit change will always create an even parity codeword. A multiple fault which causes an even number of bits to change at the output is not detectable since it creates an incorrect codeword. This situation

occurs for at least one code input (in the presence of all multiple faults), so the circuit is fault secure for F_s, the set of single stuck-at faults. However, there is at least one code input which will produce a non code output (change an odd number of bits) for all multiple faults, except where all gates are affected (e.g. the output stuck at a codeword). The circuit is therefore self testing for F_t, the set of all faults affecting fewer than all k-bits. Fig. 5.10 illustrates these points for various faults. The circuit is thus TSC.

Example 5.2 : The multiplexer for two k-bit parity encoded words shown in Fig. 5.11 is also TSC. When control lines $\langle s_2 s_1 \rangle = \langle 01 \rangle$ input vector A_i becomes output vector Y_i, whilst $\langle s_2 s_1 \rangle = \langle 10 \rangle$ tranfers input vector B_i to the output. If S is the set of all odd parity k-bit vectors, the input set X is:

$$X = \{\langle s_2 s_1 A_i B_i \mid \langle s_2 s_1 \rangle = \langle 01 \rangle \text{ and } A_i \in S \text{ or } \langle s_2 s_1 \rangle = \langle 10 \rangle \text{ and } B_i \in S\} \quad (5.5)$$

When the multiplexer is set to transfer A_i to Y_i, any fault affecting the data path of each individual bit is an identical problem to the set of buffers in example 5.1. A fault in the control path of s_1 (eg. AND gate input SA0) for a particular bit will affect the data flow at the output of its input AND gate (Fig. 5.11) and is therefore equivalent to a fault in the data path. This hypothesis is also valid for a transfer of B_i to Y_i in conjunction with control line s_2, so the multiplexer is TSC for the same fault sets given in example 5.1. If s_2 and s_1 fan-out from a common source, then faults before the fan-out point are not detectable, in general, at the multiplexer output, since they could represent another valid control input, as shown in Fig. 5.12. These faults will have to be detected by some other means such as, for example, encoded control lines.

Examples 5.1 and 5.2 transfer an input vector to their

outputs without modification, so S could be any error detecting code and the circuits would still be TSC, although not neccessarily for the same faults sets. Consider now an example which combines two input codewords to produce a different output codeword. It performs a function $Z(X_i,Y_i)$ such that X_i and $Y_i \in S$ implies $Z(X_i,Y_i) \in S$, i.e. function Z preserves the encoding.

Example 5.3 : A modulo 2^k-1 ripple adder for data encoded in an AN code is described in section 4.3.5 and illustrated in Fig 5.13a. Assuming that the carry and sum bits of each full adder are computed with independent subcircuits, as shown in Fig. 5.13b, then a single fault in the circuit will have one of the following effects:

1) No effect.
2) Carry out is 1 when it should be 0, or vice versa.
3) Sum out is 1 when it should be 0, or vice versa.

Effects 2) and 3) will produce a result which is incorrect by a power of two. Clearly, if the multiplying factor of the code , A, is a power of two, then the output becomes an incorrect codeword. However, if low cost AN codes are used, where $A = 2^b-1 \mid b$ an integer > 1 (see section 4.3.5), this problem does not arise and the adder is fault secure for all single stuck-at faults. Fig. 5.14 shows the effect of these faults for A=2 and A=3. There is at least one code input combination which will produce a non code output for faults affecting fewer than all k-bits, so under these conditions the circuit is self testing and hence TSC.

5.3.2 : TSC Networks

The difficulties of providing general design rules for TSC circuits (see next Chapter) means that large TSC networks are often formed from the interconnection of a number of smaller TSC logic blocks. However, these blocks may be TSC in isolation, but when interconnected to form a large

network, they could be neither fault secure or self testing. A block may not be fault secure because a non code output may not propagate to the network outputs. In addition, the block may not be self testing because its inputs are outputs from other blocks. As a result, the required patterns to ensure self test might not exist during normal operation.

Anderson [5.2] gives a theorem for TSC networks with sufficient although not necessary conditions. Before presenting this theorem, two further definitions are required.

DEFINITION 5.12 : A block of a network is defined to be fully exercised if it receives its entire input code space B from the application of normal input set X to the network.

This definition is illustrated in Fig. 5.15.

DEFINITION 5.13 : A block of a network is defined to be securely located if, on the basis of its inputs, the preceding subnetwork of the total network is fault secure, i.e. its input encoding will detect all $f \in F_{sns}$ in the previous subnetwork, where F_{sns} is the union of F_s for each block in this subnetwork.

This definition is illustrated in Fig. 5.16.

THEOREM 5.1 : An interconnection of logic blocks is TSC for the network secure fault set F_{ns} (F_{ns} is the union of F_s for each block) and the network self testing fault set F_{nt} (F_{nt} is the union of F_t for each block), if each block is:

1) TSC (fault secure for its F_s and self testing for its F_t).
2) Code disjoint
3) Fully exercised

4) Securely located

Proof: The network must be shown to be fault secure and self testing.

Fault Security: Since each block is fault secure for its F_s, a faulty block generates either the correct codeword or a non codeword at its outputs. For a non code output, succeeding blocks will not receive an incorrect code input as they are securely located for F_{sns}. They, in turn, will produce non code outputs because they are code disjoint. Overall, a non code output from a faulty block propagates to the final output of the network as a non codeword. Hence the network is fault secure.

Self test: Since each block is self testing for its F_t and fully exercised, a faulty block will always produce a non code output for some network input. By the same argument given above, the non code output always propagates to the output of the network as a non codeword. Hence the network is self testing.

The most difficult condition to satisfy is the secure location of each block. Anderson [5.2] give two corollaries to achieve this for specific fault classes. These are given here without proof.

Corollary 5.1 : A network consisting of an interconnection of blocks is TSC for single faults ($F_{ns} = F_{nt}$) if the network has no reconvergent fan-out of block outputs and each block is i) TSC for single faults, ii) code disjoint and iii) fully exercised.

Corollary 5.2 : A network consisting of an interconnection of blocks is TSC for unidirectional faults if the network contains no inverters and each block is i) TSC for unidirectional faults, ii) code disjoint, iii) fully exercised and its inputs are encoded in a unidirectional error detecting code.

5.4 : TOTALLY SELF CHECKING CHECKERS

5.4.1 : General Structure

The output of a self checking circuit is monitored by a checker which indicates the presence of non codewords. The output from the checker could be a 0 for codewords and a 1 for non codewords, as indicated in Fig. 5.2. However, the checker needs to be totally self checking so that faults within it can be detected just as easily as those in the circuit it is monitoring. It must not allow a detectable error (non codeword) in the circuit it is monitoring to be ignored because of a failure within itself. Its output must therefore be encoded in an error detecting code S_c. The encoding suggested above is clearly not an error detecting code, as a single fault, the output SA0 (no error), would never be detected. Suitable error detecting codes have been mentioned in section 4.3.3. These are the duplication code, S = {⟨00⟩, ⟨11⟩}, and the 1-out-of-2 code, S = {⟨01⟩,⟨10⟩}. The 1-out-of-2 code is preferred since it detects unidirectional multiple errors (see section 3.1.4), such as those produced by the loss of power, and this is the code which has been generally adopted.

Lines constantly at one logic level are thus not allowed in a network which is to be self testing, since the lines would not be tested for being stuck at that value. This must also be true of the checker output, even if it is encoded. If a failure (or failures) within the checker caused its output to be permanently ⟨10⟩, for example, then this is just as undetectable as the single line SA0 above. Therefore, the checker output must alternate between ⟨01⟩ and ⟨10⟩ during normal operation to indicate an error free network. This means that there must be at least one code input to the checker which is mapped to each of these outputs.

The system where 0 is represented as ‹01› or ‹10› and 1 represented as ‹11› or ‹00› (or vice versa) is also known as Morphic logic, where Morphic Boolean functions replace the standard functions of OR, AND and NOT [5.4,5.5].

Anderson [5.2] presented the model given in Fig. 5.17 for a totally self checking network, which consists of a functional circuit and a checker which are both TSC. The functional circuit has a normal input set X_f and output set S_f, whilst the checker has a normal input set X_c and output set S_c. In addition:

$$X_f \subseteq \{0,1\}^n \tag{5.6}$$
$$S_f \subset \{0,1\}^m \tag{5.7}$$
$$X_c = S_f \tag{5.8}$$
$$S_c = \{\langle 01\rangle, \langle 10\rangle\} \tag{5.9}$$

The fault free output function of the checker must map code inputs to code outputs and non code inputs to non code outputs, so it must be code disjoint.

It is interesting to note that the checker does not necessarily need to be fault secure. It must certainly be self testing and code disjoint, but if a fault produces an incorrect output codeword this is not a problem, since as an error indicator, the output of the checker is either a codeword (fault free) or non codeword (error). The actual codeword is not important.

Applying Theorem 5.1 to Fig. 5.17 demonstrates that the network is TSC. Its secure and tested fault sets are the unions of corresponding fault sets for functional circuit and checker. The normal input set of the network is X_f and the output code set S_n is defined as;

$$S_n = S_c \times \{0,1\}^m \tag{5.10}$$

This definition allows errors to be detected from an observation of the checker output only. If the output of

the network was specified as S_c alone, then network fault security would not guarantee that a code space output from the checker implied a correct functional circuit, see example 5.6. During fault free operation the output of the network is $S_c \times S_f$.

Chapter 6 describes the design of several TSC checkers.

5.4.2 : TSC Checker for Separable Codes

If the error detecting code is separable, then a checker for that code will consist of an equality checker and a check bit generator, as shown in Fig. 5.18. The equality checker compares the check bits of the codeword with new check bits generated from the data part of the codeword. The following theorem demonstrates that a checker is TSC if the equality checker is TSC:

THEOREM 5.2 : For the conditions:

i) A separable error detecting code S with codewords $\langle C_i, X_i \rangle$, where $X_i \in X$ is the original data and $C_i \in C$ the check bits generated from $C_i = F(X_i)$.
ii) The checking network consists of a check bit generator which computes $F(X_i)$ and an equality checker which compares C_i and $F(X_i)$ (see Fig. 5.18).

The network is a TSC checker for S if the equality checker is TSC for the duplication code S', where:
$S' = \{\langle C_i, C_i \rangle \mid \exists\ X_i \in X \text{ such that } F(X_i) = C_i\}$.

The proof for this theorem is given by Wakerly in [5.6]. The equality checker has to be TSC for S' to satisfy the above theorem. It has two sets of C_i as inputs, which form S'. In most cases C_i will take on all possible combinations, $C_i \in C$ and $C \subseteq U$, so the equality checker will automatically be TSC. However, if the check bits do not take on all possible values, as for the residue codes illustrated in Fig. 5.19, then the self testing property

must be verified for those that do. The theorem also depends on the data part of S taking on all possible values to completely test the check bit generator for faults which affect its operation. Again, if not all these occur, then the structure of the check bit generator must be examined to verify its self testing property for those which do.

Overall the problem of designing TSC checkers for separable codes can be reduced to the much simpler problem of designing check bit generators for these codes (see examples in section 4.3), with no consideration of their self checking properties.

5.5 : PARTIALLY SELF CHECKING NETWORKS

If a logical operation is performed on two codewords the result is not, in general, a codeword; i.e. the code is not preserved. If the encoding is separable (separate data and check bits), then a logical operation performed on the two data parts will be correct, but the same operation performed on the two sets of check bits will not, in general. Duplication, however, is one code that is preserved by logical operations, but of course has a high redundancy.

A solution is to calculate new check bits from the data output after the logical operation. This has previously been referred to in sections 4.3.1 and 4.3.4 as check bit prediction. This is a practical scheme if it can be implemented in a self checking manner at low cost. Partially self checking (PSC) networks fulfill this requirement. They achieve less redundancy by not requiring the immediate detection of errors.

Wakerly [5.6,5.7] has proposed three forms of PSC network, which are detailed below.

5.5.1 : Type 1 Networks

The simplest PSC network is referred to as the type 1 model and shown in Fig. 5.20. It uses a TSC functional circuit with a fault free output function which maps the normal input set, $X_f \subseteq \{0,1\}^n$ to an output code space, $S_f \subset \{0,1\}^m$. The TSC checker has normal input set $X_c = S_f$ and output code space $S_c = \{\langle 01\rangle, \langle 10\rangle\}$.

There are also two control gates which enable or disable the error indication from the network. With control lines $\langle s_2 s_1\rangle$ set to $\langle 01\rangle$, the network error indicator mimics the the checker outputs, but when the lines are set to $\langle 10\rangle$, they force the error indicator to $\langle 10\rangle$ (no error).

The output code space for a type 1 PSC network is $S_c \times S_f$ as with a TSC network. The normal input set of the network is vectors of the form $\langle s_2 s_1 X_i\rangle$, where X_i is the functional circuit input. When functional circuit inputs from X_f are expected, $\langle s_2 s_1\rangle$ is set to $\langle 01\rangle$ and the network is logically equivalent to the TSC network of Fig. 5.17. However, when inputs are not from X_f, the functional circuit output may not be a codeword, so $\langle s_2 s_1\rangle$ is set to $\langle 10\rangle$ which disables the checker. When used in this manner the network of Fig. 5.20 is PSC.

Let F_a be the set of all single stuck-at faults on the control gates. The secure and tested fault sets of the network are the unions of F_a with the corresponding fault sets of functional circuit and checker. The secure input set of the network, N_n, is:

$$N_n = \{\langle s_2 s_1 X_i\rangle \mid (\langle s_2 s_1\rangle = \langle 01\rangle) \text{ and } (X_i \in X_f)\} \qquad (5.11)$$

The network may also be operated with insecure inputs from the set N_n':

$$N_n' = \{\langle s_2 s_1 X_i\rangle \mid (\langle s_2 s_1\rangle = \langle 10\rangle) \text{ and } (X_i \notin X_f)\} \qquad (5.12)$$

Normally during its insecure mode of operation the func-

tional circuit will only receive inputs outside X_f, but, in practice, any input could occur, so N_n' is better defined as:

$$N_n' = \{\langle s_2 s_1 X_i \rangle | (\langle s_2 s_1 \rangle = \langle 10 \rangle)\} \qquad (5.13)$$

A type 1 PSC network alternates between secure and insecure modes of operation. The normal input set X_n of the network is thus:

$$X_n = N_n \cup N_n' \qquad (5.14)$$

THEOREM 5.3 : The network shown in Fig. 5.20 and described above is PSC.

Proof : During its secure mode of operation, the network is self testing and fault secure for the fault sets of the functional circuit and checker (both TSC networks). Therefore, the network is also self testing with inputs from X_n since $X_n \supset N_n$. This leaves the evaluation of self test and fault security for F_a.

Self test : All faults except d SA0 and a SA1 are tested by some input from N_n. The faults d SA0 and a SA1 are not detected since d=0 and a=1 during the secure mode (constant lines), but they will be detected by some input from N_n', since each changes the correct error indicator output from ⟨10⟩ to ⟨11⟩ or ⟨00⟩. Thus all faults in F_a are tested by some input from X_n.

Fault security : A single fault from F_a causes, at most, a single bit change in the error indicator output, producing either the correct output or a non codeword.

Example 5.4 : A type 1 PSC network is the n-bit parity checked bus buffer shown in Fig. 5.21. The TSC functional circuit is the n-bit bus buffer of Fig 5.9. The TSC parity checker is based on a design to be described in Chapter 6. The control inputs $\langle s_2 s_1 \rangle$ are set to ⟨01⟩ for

the transmission of odd parity words and to ⟨10⟩ for words of unknown parity.

The application of type 1 networks is limited, since a re-encoded functional output is not available during their insecure mode. However, in the example of Fig. 5.21, re-encoded parity is always available from the parity checker at line P. Type 2 networks implement this concept.

5.5.2 : Type 2 Networks

A type 2 network is a type 1 PSC network which uses the TSC checker design described in section 5.4.2 and whose functional circuit output is a separable code. Fig. 5.22 shows such a network.

A re-encoded functional output is provided by the check bit generator within the checker. The input sets, fault sets and output code space are the same as those for a corresponding type 1 network. Thus, ignoring the re-encoded functional output, type 2 networks are type 1 networks with a specific TSC checker and hence they are PSC.

A non codeword output caused by a fault in the re-encoding operation is detected by the equality checker. These results are summarised in the following theorem.

THEOREM 5.4 : A type 2 network described above and illustrated in Fig. 5.22 is PSC. In the absence of faults the re-encoded functional circuit output is always a codeword.

5.5.3 : Type 3 Networks

A disadvanatge of type 2 networks is that the functional circuit output is delayed by the check bit generator in the re-encoding process. In a TSC network or a type 1 PSC network, the functional output delay is due to the

functional circuit alone, whereas in a type 2 PSC network it is the sum of functional circuit and check bit generator delays. During its insecure mode, the re-encoding process in any PSC network will always introduce delay, but a type 3 network reduces this delay to that of a multiplexer during the secure mode.

Fig. 5.23 shows a type 3 network, consisting of a TSC functional circuit, a TSC equality checker and multiplexer which selects either the check bits of the functional circuit, or the output of the check generator as the network check bits.

The equality checker compares the network check bits with the generated check bits. When $\langle s_2 s_1 \rangle$ is $\langle 01 \rangle$ (secure mode) the network is logically equivalent to a TSC network. When $\langle s_2 s_1 \rangle$ is $\langle 10 \rangle$ (insecure mode) the functional circuit output is re-encoded and the equality checker compares the generated check bits with themselves, producing a good output.

The normal input set, secure input set and output code space of a type 3 network are the same as those of a type 1 or 2 PSC network. Let F_a be the set of all single faults affecting the control gates except u_j SA1 (see Fig. 5.23). The secure and tested fault sets of the network are the unions of F_a with the corresponding fault sets of the functional circuit and checker.

THEOREM 5.5 : A type 3 network described above and illustrated in Fig 5.23 is PSC.

The proof for this theorem is similar to that for theorem 5.3 and requires a demonstration of the self testing and fault secure capabilities of the network for F_a.

Although type 3 networks avoid the delay associated with a re-encoded functional output during their insecure mode, they have two disadvanatges:

1) They require more gates than a type 2 network with a corresponding increase in cost.
2) They have a set of single stuck-at faults for which the network is generally not self testing or fault secure (u_j SA1).

Wakerly [5.6], however, shows how to overcome the latter problem.

5.5.4 : A Partially Self Checking Logic Unit

Example 5.5 : The bit-slice (see section 5.7) of Fig. 5.24a can perform sixteen different Boolean functions on two input variables a_j and b_j, dependent on the setting of control inputs s_4, s_3, s_2 and s_1. The functions available are detailed in Fig. 5.24b. The circuit may be duplicated to form a bit-sliced functional circuit which can perform any of these operations on two input vectors A_i and B_i such that for each value of $r = \langle s_4 s_3 s_2 s_1 \rangle$ the circuit performs a function $F_r(A_i,B_i)$. If A_i and B_i are encoded in an error detecting code S with a Hamming distance of two, and if a function preserves that encoding ($A_i,B_i \in S$ and $F_r(A_i,B_i) \in S$), then the functional circuit is fault secure for all single bit-slice faults from theorem 5.7. The secure input set N_r for $F_r(A_i,B_i)$ is:

$$N_r=\{\langle s_4 s_3 s_2 s_1 A_i B_i\rangle \mid \langle s_4 s_3 s_2 s_1\rangle = r \text{ and } A_i,B_i \in S\} \qquad (5.15)$$

If the encoding is preserved by a number of r, say $r \in R$, then the secure input set N of the circuit is:

$$N = \bigcup_{r \in R} N_r \qquad (5.16)$$

Consider S to be the set of even parity n-bit vectors, where n is even. This encoding is preserved by operations $A_i \oplus B_i$, $\overline{A_i \oplus B_i}$, A_i, B_i, $\overline{A_i}$, $\overline{B_i}$, 0 and 1 (A_i, $B_i \in S$). Fig. 5.25 details the single stuck-at faults in the bit-

lice (Fig 5.24a) which are detected by these operations [5.7]. Inspection of Fig. 5.25 reveals that all single stuck-at faults are detected if both $A_i \oplus B_i$ and $\overline{A_i \oplus B_i}$ are used. The same is true if all four operations A_i, B_i, $\overline{A_i}$ and $\overline{B_i}$ are used. If the functional circuit has an input set which contains any such set of code preserving operations then the circuit is self testing. The circuit is also fault secure for code preserving operations, so under these conditions it is TSC (secure mode). It can therefore be used in a partially self checking network which re-encodes the output for those functions which are not code preserving.

The 74181 arithmetic logic unit/function generator [5.8] contains four circuits similar to that in Fig. 5.24a. It can perform the logic functions of Fig. 5.24b or a set of arithmetic functions. Additional carry logic means that it is not a bit-sliced circuit, but if A_i and B_i are encoded in an arithmetic error detecting code which has a Hamming distance of two, then their encoding is preserved by the addition and subtraction operations. These functions will test the carry logic, and when combined with code preserving functions which test all stuck-at faults, will allow the 74181 to be TSC [5.7]. This can then form the basis of a PSC arithmetic logic unit.

5.6 : SELF TESTING ONLY CIRCUITS AND NETWORKS

TSC and PSC circuits are self testing for a set of inputs X and fault secure for a non-null subset N of X. N is a null set for self testing only (STO) circuits.

Example 5.6 : Consider the 2 to 4 line decoder shown in Fig. 5.26. A 2-bit input vector $X_i=\langle x_2 x_1 \rangle$ is translated into a 1-out-of-4 coded vector $Y_i=\langle y_4 y_3 y_2 y_1 \rangle$. The error indicator $E_i=\langle e_2 e_1 \rangle$ is formed by ORing the outputs into two partitions:

$$e_1 = y_4 + y_1 \,, \qquad (5.17)$$

$$e_2 = y_3 + y_2 \tag{5.18}$$

In the absence of faults $E_i \in \{\langle 01\rangle,\langle 10\rangle\}$. The normal input set is the set of 2-bit vectors $X_i \in \{0,1\}^2$, whilst from (5.10) the output code space is the set of 10-bit vectors Y_iE_i:

$$Y_iE_i \in \{0,1\}^4 \times \{\langle 01\rangle,\langle 10\rangle\} \tag{5.19}$$

where Y_i during fault free operation is:

$$Y_i \in \{y_4y_3y_2y_1 \mid y_j \in \{0,1\},\ y_4 + y_3 + y_2 + y_1 = 1\} \quad \text{('+' is plus)} \tag{5.20}$$

As usual, errors may be detected from observation of E_i alone.

Fig. 5.27 details the effect on Y_i and E_i of every single stuck-at fault in the circuit. Conclusions from this are:

1) The decoder alone is self testing and fault secure for all single stuck-at faults. It is also trivially code disjoint.
2) The checker alone is self testing and fault secure for all single stuck-at faults. It is not, however, code disjoint.
3) The overall network is self testing only for all single stuck-at faults. The secure fault set is null because a decoder output SA1 can produce a non code-word which goes undetected. This occurs when $y_j = 1$ and y_k is SA1, $j \neq k$, where y_j is in the same output partition as y_k, see Fig. 5.27.

If the output of the network was specified as E_i alone, then the fault in 3) would not be a problem, since the checker output is the same with and without the fault. However, using (5.19) means that this fault produces an incorrect codeword output from the network.

The network is TSC though for all stuck-at-0 faults. This may be acceptable if the circuit is implemented with gates whose only failure is SA0. Note that faults in x_1 and x_2 are not detectable since they create a different normal input. If it is neccessary to check these, then they need to be encoded [5.5].

Self testing only networks are formed by the interconnection of STO circuits, as shown in Fig. 5.28. The output of each STO block is either connected to the input of another block, is part of the network output, or both. Blocks which are part of the network output are termed output blocks (T_5 and T_7 in Fig. 5.28). The output code set of the network is the cross product of output block code spaces. The tested fault set of the network is the union of the fault set for each block. Sufficient conditions are given below for a network to be STO.

THEOREM 5.6 : A network which is an interconnection of logic blocks T_j is self testing only for a fault set $F_t = \bigcup_j F_j$ with normal input set X if:

i) Each block is self testing for fault set F_j with normal input set X_j, such that each input in X_j occurs for some input in X.
ii) Each block with inputs which are not network inputs or outputs is code disjoint.

Proof : It must be shown that there is a test for every fault in $F_t = \bigcup_j F_j$.

For any j and any fault f in F_j there is a network input in X such that the output of T_j is a non codeword from condition i). If the output of T_j is a network output, then the network output is also a non codeword and f tested. Otherwise the output of T_j must feed the input of another block, T_k , which is code disjoint from condition ii). Then the non code output from T_j produces a non code output from T_k. This process continues until a non code

network output is produced. In the example of Fig. 5.28, only blocks T_4, T_5 and T_7 need to be code disjoint.

5.7 : SELF CHECKING PROPERTIES OF BIT AND BYTE-SLICED CIRCUITS

A bit-sliced circuit is defined as a multi-output combinational circuit, in which each output bit is computed by an independent subcircuit, called a bit-slice. A byte-sliced circuit is similarly defined as a multi-output combinational circuit, in which each output byte is computed by an independent byte-slice. As microprocessor based systems are bus orientated, their buses are particularly suited to bit or byte-slice processing. In order to provide a more general means of analysing the self checking properties, it is convenient to consider bit and byte-sliced circuits, because of their structure and their application.

5.7.1 : Bit-Sliced Circuits

The circuits of Fig. 5.9 and 5.11 are bit-sliced circuits, whereas the circuit of Fig. 5.13a is not. Remember that a circuit is self checking if it self testing for a fault set F_t with input set X and fault secure for a fault set F_s with input set N.

Fault Security : The fault security of a bit-sliced circuit is demonstrated by the following theorem.

THEOREM 5.7 : If S is an error detecting code with a minimum Hamming distance of two, B a bit-sliced circuit with fault free output function $Z(N_i)$, such that Z maps the input set N to S, and F_s the set of all faults which affects only a single bit-slice, then B is fault secure for F_s with N.

Proof: Any fault f in F_s affects only a single bit-slice and therefore only a single output. For an input N_i, f

may or may not change that output bit. If it does not $Z^f(N_i)=Z(N_i) \in S$, but if it does, then the output is one bit different to the correct codeword and $Z^f(N_i) \notin S$.

Thus, a bit-sliced circuit is fault secure for all single bit-slice faults with all inputs N_i that are mapped to codeword outputs. The checker must be disabled for inputs outside N during normal operation. This is therefore a PSC network.

Self Test: The self testing property for all single faults depends on the bit-slice structure and normal input set X. However, the self testing capability of a bit-sliced circuit can be determined from the individual slices.

Consider a single bit-slice B_j within a bit-sliced circuit, as shown in Fig. 5.29. The input to B_j is A_{ij}, so that its output is $Z_j(A_{ij})$. Given X, a set of input vectors A_j to the slice B_j can be determined, see Fig. 5.29.

Example 5.7 : The TSC multiplexer shown in Fig. 5.30 has an input vector $\langle s_2 s_1 a_j b_j \rangle$ to each bit-slice. During normal operation $\langle s_2 s_1 \rangle$ is set to $\langle 01 \rangle$ or $\langle 10 \rangle$, so that for every j, a_j and b_j should take on all four possible combinations. The set A_j for B_j is thus:

$$A_j = \{\langle 0100\rangle,\langle 0101\rangle,\langle 0110\rangle,\langle 0111\rangle, \langle 1000\rangle,\langle 1001\rangle,\langle 1010\rangle,\langle 1011\rangle\} \quad (5.21)$$

Given A_j, then the tested fault set for the complete bit-sliced circuit can be determined.

THEOREM 5.8 : If the output of a bit-sliced circuit is encoded in an error detecting code S, where S has a minimum Hamming distance of two, and for each bit-slice B_j there is a set of faults F_{tj}, such that for an input A_{ij} in A_j, $Z_j^f(A_{ij}) \neq Z_j(A_{ij})$, then the bit-sliced circuit is self testing for the fault set $F_t = \bigcup_j F_{tj}$.

Proof : For any fault f in F_{tj}, there is a bit-slice input A_{ij} in A_j due to circuit input X_i in X such that $Z_j^f(A_{ij}) \neq Z_j(A_{ij})$. No other bit is affected by f, so the circuit output $Z^f(X_i)$ is different from $Z(X_i) \in S$ by a single bit and therefore $\notin S$. The circuit is thus self testing for any fault in F_j and self testing for any fault in $F_t = \bigcup_j F_{tj}$.

The self testing ability of a bit-sliced circuit can be determined by considering each bit-slice separately. In many cases the bit slices B_j are identical (for example the multiplexer of Fig. 5.30), as are the input sets A_j and fault sets F_{tj}. A standard set of faults F_t together with a standard set of inputs A from X can therefore be ascertained. F_t is determined by methods which identify the set of faults detected for a particular test A_i in A. Alternatively, F_t is deemed to be a known fault set and every A_i in A examined to prove that every f in F_t is tested.

5.7.2 : Byte-Sliced Circuits

The results of theorems 5.7 and 5.8 for bit-sliced circuits can be extended to cover byte-sliced circuits with outputs encoded in a byte error detecting code. Such circuits are fault secure for all single byte faults. As with bit-slice circuits, their self testing capabilities are dependent on structure and normal input set.

Example 5.8 : Consider the byte-slice multiplexer for 8-bit words shown in Fig. 5.31. The select input, s, determines whether A_i or B_i is transfered to the output. The transfer takes place when the enable input is at 0, otherwise an all 0 output occurs. If the input vectors are encoded in a byte error detecting code for 4-bit bytes such that the all zero vector (all outputs at 0) is a codeword, then the circuit is fault secure for all single byte faults. This statement is independent of the struc-

of the quad 2 to 1 multiplexer. Examples of suitable byte error detecting codes are the b-adjacent code shown in Fig. 4.10, or the checksum code shown in Fig. 4.12. Self test is not independent of multiplexer structure so each individual byte has to be examined to determine the level of self test.

5.8 : SELF CHECKING SEQUENTIAL CIRCUITS

A model of a sequential machine is required to develop the definitions of self test and fault security given for sequential circuits at the beginning of the Chapter. The model depicted in Fig. 5.32 will be used which is specified by $\langle V,Q,W,\delta,\omega\rangle$, where:

V is the set if input vectors.
Q is the set of states.
W is the set of output vectors.
δ is the next state function of the fault free machine, $\delta : V \times Q \rightarrow Q'$.
ω is the output function of the fault free machine, $\omega : Q \rightarrow W$.

V contains all possible binary input vectors, W all possible output vectors and Q all possible states, regardless of whether they occur in normal operation. For fault detection purposes, the output of the circuit must be encoded in an error detection code, such that $S \subset W$ and non codewords detected. It is assumed that only the circuit output is observable; i.e. the state and input information is not observable, unless it is part of the output function.

5.8.1 : Fault Security

A sequential circuit fault may change the next state function, the output function, or both. The next state function of a machine in the presence of a fault is denoted by δ^f and the output function by ω^f. Fault

security requires that output function failures must produce either the correct code output or a non code output. Next state failures must produce either the correct state or a non code output before an incorrect next state produces an incorrect code output. It is therefore possible for a sequential machine to be in several incorrect states before a failure is indicated, as long as none of these incorrect states produces an incorrect code output. In practice, however, a transition to an incorrect state needs to be indicated immediately. This requires the set of states occupied by the fault free machine to be encoded in an error detecting code, $P \subset Q$, and also requires the output function to be a code disjoint mapping from P to S.

DEFINITION 5.14 : A sequential circuit is fault secure for a fault set F_s with input set N ($N \subseteq V$), if, $\forall N_i \in N$, $\forall f \in F_s$ and $\forall P_i \in P$, either :

i) $\delta^f(N_i,P_i) = \delta(N_i,P_i)$ and $\omega^f(\delta^f(N_i^-,P_i^-)) = \omega(\delta(N_i^-,P_i^-))$

or

ii) $\omega^f(\delta^f(N_i^-,P_i^-)) \notin S$

This definition, illustrated in Fig. 5.33, requires that all faults either produce no effect on the next state and output function for a particular input, or produce an immediate error indication. Visiting several error states whilst maintaining the correct output before a detectable error is not allowed. (Diaz, however, in his definition could allow this behaviour [5.9].) The above definition also requires that a fault affecting both the next state and output functions must still give an error indication. When logic is shared between both functions this error indication is difficult to guarantee, so shared logic must be avoided. The definition of fault security does not require an error indication to be maintained; i.e. once a faulty circuit produces a non code state and indicates an error, the definition does not prevent it from returning to an incorrect code state. Many sequential machine

designs prevent this behaviour by 'error trapping', whereby the machine maintains a non code state if one is entered.

The general block diagram of Fig. 5.32 may be modified to give the fault secure machine in Fig. 5.34. The current state P_i is held in a state register implemented with (say) D type flip-flops fed by a common clock signal. It is assumed that the clock signal is checked separately, see Chapter 6. The state register is updated with the output of a circuit which computes $\delta(N_i,P_i)$ and the machine output is produced by $\omega(P_i)$. The state vector P_i should be from an error detecting code P so that the state register is fault secure. If, for example, P is a single bit parity code, then the state register is fault secure for all faults which affect a single bit. The next state circuit must also be fault secure. This is accomplished by using an independent subcircuit for each bit. Finally, the output circuit should be a fault secure, code disjoint mapping from state code P to output code S, so that $P_i \in P$ becomes $S_i \in S$.

An m-out-of-n code is equally suitable for encoding the states of a fault secure machine. This has the advantage of detecting undirectional errors.

Example 5.9 : Consider the state diagram, state table and logic diagram given in Fig. 5.35 for a synchronous sequential machine. Output y is set when three consecutive 1's are received by input x and reset by the next 0 at x thereafter. The design has four states and requires two flip-flops. A fault secure version of the machine is achieved by encoding the states in a single parity code and encoding the output in a 1-out-of-2 code, as shown in Fig. 5.36. The machine still has four states, but now requires three flip-flops because of the additional parity 'bit' and a second output to support the 1-out-of-2 encoding. The machine is now fault secure for a single gate or flip-flop fault.

5.8.2 : Self Test

A sequential machine is self testing if every fault is tested during normal operation. If $P \subset Q$ is the set of states during normal operation and $X \subseteq V$ is the set of normal inputs, then one way to define self test would be the existence of a state P_i in P and an input X_i in X for each fault f in F_t, such that $\omega^f(\delta^f(X_i^-,P_i^-)) \notin S$ [5.9]. However, this definition is not adequate since the inputs and states of the machine may be correlated and there is no guarantee that a particular input/state pair occurs in practice. For example, input strings with only two or three consecutive 1's will not visit the states marked with a '*' in Fig. 5.36. Therefore, the set XP of normal input/state pairs has to be defined, where $\langle X_i,P_i \rangle$ is in XP if X_i occurs with state P_i during normal operation. This allows a more satisfactory definition of self testing sequential circuits to be made [5.6].

DEFINITION 5.15 : A sequential circuit is self testing for a normal input/state set XP and fault set F_t, if, $\forall f \in F_t$, $\exists \langle X_i,P_i \rangle \in XP$, such that $\omega^f(\delta^f(X_i^-,P_i^-) \notin S$.

This definition is illustrated in Fig. 5.37. The sequential machine in Fig. 5.34 is self testing if the next state function, state register and output function are all tested by normal inputs. Self testing is not as easy to achieve as fault security. The design of example 5.9 in Fig.5.36, for instance, is not self testing for SA0 faults on the inputs of its 3-input AND gate, even if all input/state pairs occur. Requirements for the self testing of seqential machines are discussed by Diaz [5.9,5.10] and Ozguner [5.11].

5.9 : LITERATURE REVIEW

Carter and Schneider laid the foundations for self testing circuit theory in their historic 1968 paper [5.1], with a

model for dynamically checked logic. They also presented designs for parity and 1-out-of-2 code checkers, which have since been universally adopted as a basis for TSC checkers. Carter in conjunction with other IBM personnel subsequently expanded these ideas from the view point of fault tolerant computing [5.12-5.14] and Morphic Boolean functions [5.4]. Carter summarised many of these concepts in his excellent 1974 paper [5.5]. The most recent documentation from the team has been a fault tolerant memory system [5.15] and a self checking computer design [5.16]. The latter is a significant development from their first paper.

In his thesis, Anderson [5.2] formalised the initial ideas of Carter, with definintions for TSC circuits and networks, plus a discussion on how to test them. He also presented an important design technique for TSC m-out-of-n checkers, which was extended in a later paper [5.17]. Many papers have been written on the design of TSC code checkers and these will be detailed in the next Chapter. Kolupaev [5.18] has more recently considered TSC networks, with modifications to the fault secure and self testing definitions proposed by Anderson.

Using a more mathematical approach, Smith [5.19], in his thesis,detailed TSC circuits for single and unidirectional faults, as well as considering their behaviour for unmodelled faults. A subset of this work appeared in [5.20], with further papers discussing strongly fault secure logic networks [5.21] and TSC system design [5.22]. Self checking systems are the subject of Chapter 7, prompted by the work of Moreira de Souza et al [5.23]. Strongly fault secure systems are also considered by Nanya [5.24], whilst a design technique for TSC circuits with unrestricted stuck-at faults using redundancy in time and space has been proposed by Rao [5.25].

In 1974 Wakerly introduced the concept of PSC circuits and networks [5.7], which subsequently formed part of his book

[5.6]. In the book he also used modified definitions of the fault secure and self testing properties. The reliability of PSC circuits has been discussed by Gay [5.26] and a signal reliability evaluation of self checking networks proposed by Hong [5.27]

The majority of the papers referenced so far are concerned with combinational circuits only, but Diaz has presented a number of papers on the design of sequential TSC circuits [5.9,5.10,5.28], along with Smith [5.22], Nanya [5.29], Viaud [5.30] and Ozguner [5.11]. The code disjoint property has been discussed by Jansch [5.31] for sequential circuits and by Nicolaidis [5.32] for combinational circuits.

The construction of TSC circuits using programmable logic arrays (PLAs) has been widely investigated [5.33,5.34], particularly for checkers (see Chapter 6), whilst the specific problems associated with TSC circuit implementation in technologies such as MOS [5.35,5.36], CMOS [5.37] and I^2L [5.38,5.39] have also been examined.

Specific examples of self checking circuits include a self checking form of the linear feedback shift register (mentioned many times in Chapter 4), developed by Lu [5.40], a self checking periodic signal checker, proposed by Usas [5.41] and a self testing decoder described by Carter [5.42].

All of the above papers detail various aspects of self checking hardware, but self checking software also needs to be considered [5.43].

5.10 : REFERENCES

5.1) Design of dynamically checked computers. - W. C. Carter, P. R. Schneider; Proc. IFIPS Congress, Vol. 2; Edinburgh; August 1968; North-Holland Publishing Company, Amsterdam (1969); pp. 878-883.

5.2) Design of self-checking digital networks using coding techniques - D. A. Anderson; Coordinated Science Laboratory Report R-527; September, 1971; University of Illinois, Urbana, Illinois, USA.

5.3) The error latency of a fault in a combinational digital circuit - J. J. Shedletsky, E. J. McCluskey; FTCS-5*; pp. 210-214.

5.4) Computer error control by testable morphic Boolean functions - A way of removing hardcore - W. C. Carter, A. B. Wadia, D. C. Jessep; FTCS-2*; pp. 154-159.

5.5) Theory and use of checking circuits - W. C. Carter; Infotech*; pp. 415-454

5.6) Error detecting codes, self checking circuits and applications - J. F. Wakerly: Elsevier - North Holland; New York; 1978; Chapter 3, pp. 54-99.

5.7) Partially self-checking circuits and their use in performing logical operations - J. F. Wakerly; IEEE Trans. Comput.; Vol. C-23, No. 7; July, 1974; pp. 658-666.

5.8) The TTL book for design engineers - The engineering staff of Texas Instruments components group; Texas Instruments; Fourth European edition; 1980; pp. 7:271-7:281.

5.9) Design of totally self-checking and fail-safe sequential machines - M. Diaz; FTCS-4*; pp. 19-24.

5.10) Unified design of self-checking and fail-safe combinational circuits and sequential machines - M. Diaz, P. Azema, J. M. Ayache; IEEE Trans. Comput.; Vol. C-28, No. 3; March, 1979; pp. 276-281.

5.11) Design of totally self-checking asynchronous and synchronous sequential machines - F. Ozguner; FTCS-7*; pp. 124-129.

5.12) Logic design for dynamic and interactive recovery - W. C. Carter, D. C. Jessep, A. B. Wadia, P. R. Schneider, W. G. Bouricius; IEEE Trans. Comput.; Vol. C-20, No. 11; November, 1971; pp. 1300-1305.

5.13) A theory of design of fault-tolerant computers using standby sparing - W. C. Carter, W. G. Bouricius, D. C. Jessep, J. P. Roth, P. R. Schneider, A.

B. Wadia; FTCS-1*; pp. 83-86.

5.14) A theory of design of fault-tolerant computers using standby sparing - W. C. Carter, W. G. Bouricius, D. C. Jessep, J. P. Roth, P. R. Schneider, A. B. Wadia; IBM Research Yorktown; RC 3261; February, 1971; pp. 1-11.

5.15) Implementation of an experimental fault-tolerant memory system - W. C. Carter, C. E. McCarthy; IEEE Trans. Comput.; Vol. C-26, No. 6; June, 1976; pp. 557-568.

5.16) Cost effectiveness of self checking computer design - W. C. Carter, G. R. Putzola, A. B. Wadia, W. G. Bouricius, D. C. Jessep, E. P. Hsieh, C. J. Tan; FTCS-7*; pp. 117-123.

5.17) Design of totally self-checking check circuits for m-out-of-n codes; D. A. Anderson, G. Metz; IEEE Trans. Comput.; Vol. C-22, No. 3; March, 1973; pp. 263-269.

5.18) Cascade structure in totally self-checking networks - S. Kolupaev; FTCS-7*; pp. 150-154.

5.19) The design of totally self-checking combinational circuits - J. E. Smith; Coordinated Science LaboratoryReport R-737; August, 1976; University of Illinois, Urbana, Illinois, USA.

5.20) The design of totally self-checking combinational circuits - J. E. Smith, G. Metz;FTCS-7*;pp. 130-4.

5.21) Strongly fault secure logic networks - J. E. Smith, G. Metz; IEEE Trans. Comput.; Vol. C-27, No. 6, June, 1978; pp. 491-499.

5.22) A theory of totally self checking system design - J. E. Smith, P. Lam; IEEE Trans. Comput.; Vol. C-32, No. 9; September, 1983; pp. 831-844.

5.23) A research oriented microcomputer with built in auto-diagnostics - J. Moreira de Souza, E. Peixoto Paz, C. Landrault; FTCS-6*; pp. 3-8.

5.24) Error secure/propagating concept and its application to the design of strongly fault secure processors - T. Nanya, T. Kawamura; FTCS-15*; pp. 396-401.

5.25) Design of totally self-checking circuits with an unrestricted stuck-at fault-set using redundancy in space and time domains - K. V. S. S. P. Rao, D. Basu; IEEE Trans. Comput.; Vol. C-32, No. 5; May, 1983; pp. 464-475.

5.26) Reliability of partially self-checking circuits

- F. A. Gay; FTCS-7*; pp. 135-142.

5.27) Signal reliability evaluation of self checking networks - K. K. Hong, Y. Thoma; FTCS-10*; pp. 257-262.

5.28) Design of self-checking microprogram controls - M. Diaz, J. Moreira de Souza; FTCS-5*; pp. 137-142.

5.29) Design of self-checking asynchronous sequential machines - T. Nanya, Y. Tohma;FTCS-10*;pp. 278-280.

5.30) Sequentially self-checking circuits; J. Viaud, R. David; FTCS-10*; pp. 263-268.

5.31) Strongly language disjoint checkers - I. Jansch, B. Courtois; FTCS-15*; pp. 390-395.

5.32) Strongly code disjoint checkers - M. Nicolaidis, I. Jansch, B. Courtois; FTCS-14*; pp. 16-21.

5.33) Design and application of self-testing comparators implemented with MOS PLA's - Y. Tamir, C. H. Sequin; IEEE Trans. Comput.; Vol. C-33, No. 6; June, 1984; pp. 493-506.

5.34) The design of totally self-checking circuits using programmable logic arrays - S. L. Wang, A. Avizienis; FTCS-9*; pp. 173-180.

5.35) Techniques for efficient MOS implementation of totally self-checking checkers - N. K. Jha, J. A. Abraham; FTCS-15*; pp. 430-435.

5.36) Analysis of a class of totally self-checking functions implemented in a MOS LSI general logic structure - M. W. Sievers, A. Avizienis; FTCS-11*; pp. 256-261.

5.37) Totally self-checking CMOS circuits using a hybrid relalization - N. K. Jha, J. A. Abraham; FTCS-15*; pp. 154-158.

5.38) Multivalued I^2L circuits for TSC checkers - D. Etiemble; FTCS-9*; pp. 181-184.

5.39) Threshold I^2L totally self checking circuits - Y. W. Yang, S. M. Liu, T. C. Chen; FTCS-10*; pp. 284-287.

5.40) Self-checking linear feedback shift registers - D. J. Lu; FTCS-10*; pp. 269-271.

5.41) A totally self-checking checker design for the detection of errors in periodic signals - A. M. Usas; IEEE Trans. Comput.; Vol. C-24, No.5; May, 1975; pp. 483-489.

5.42) A simple self-testing decoder checking circuit - W. C. Carter, K. A. Duke, D.C. Jessep; IEEE Trans. Comput.; November, 1971; pp. 1413-1415.

5.43) Self stabilising programs : The fault-tolerant capability of self-checking programs - A. Mili; IEEE Trans. Comput.; Vol. C-31 No. 7; July, 1982; pp. 685-689.

* see section B.5.

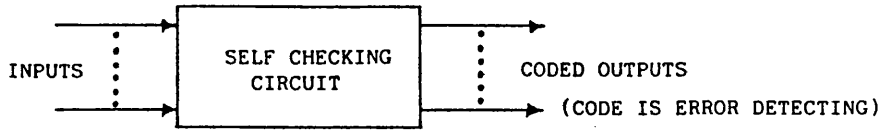


FIGURE 5.1 A SELF CHECKING CIRCUIT

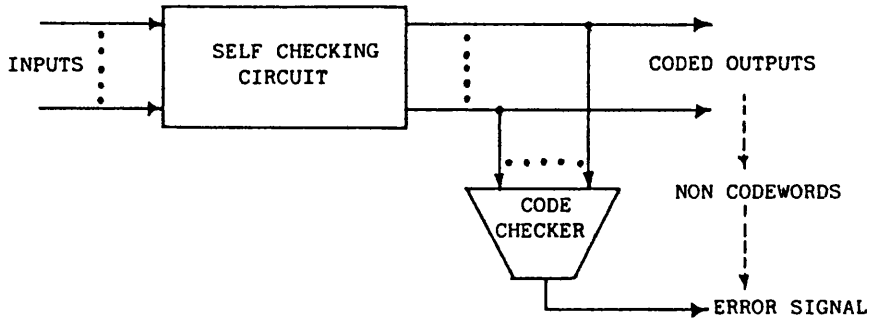


FIGURE 5.2 A SELF CHECKING CIRCUIT WITH CHECKER

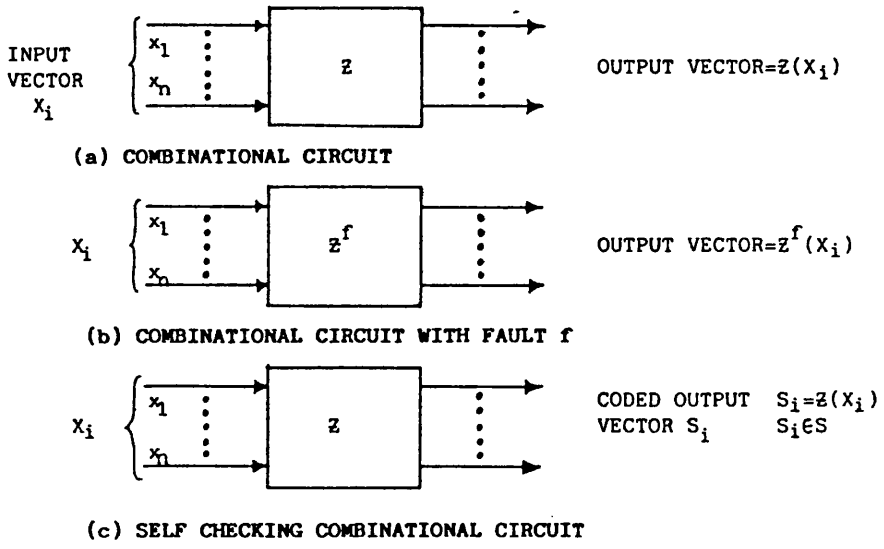


FIGURE 5.3 CIRCUIT DEFINITIONS

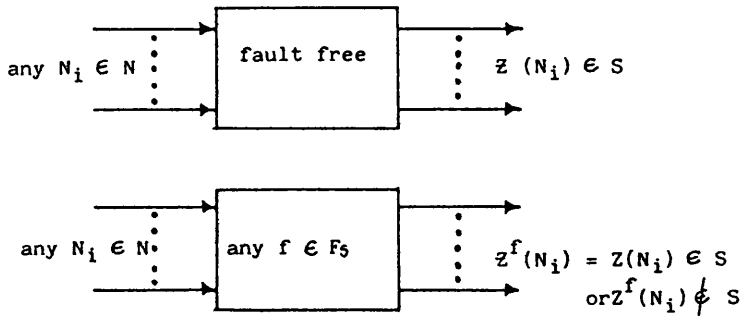


FIGURE 5.4 A FAULT SECURE CIRCUIT

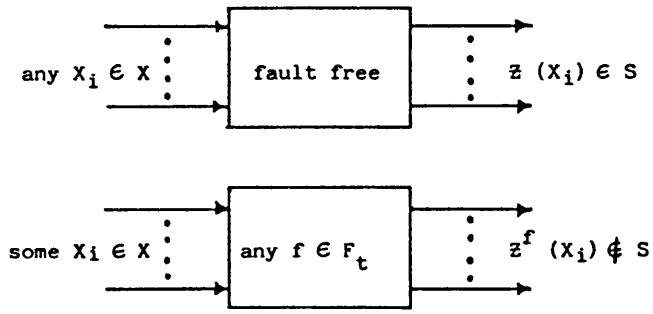


FIGURE 5.5 A SELF TESTING CIRCUIT

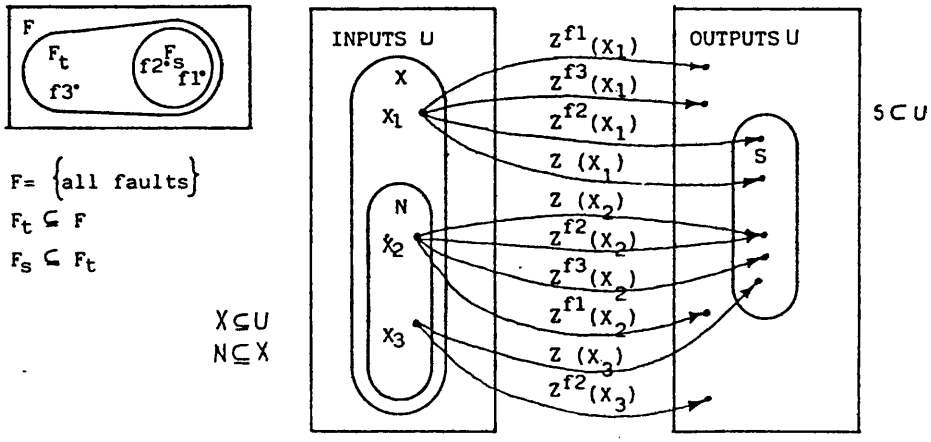


FIGURE 5.6 EXAMPLES OF SELF TEST AND FAULT SECURITY

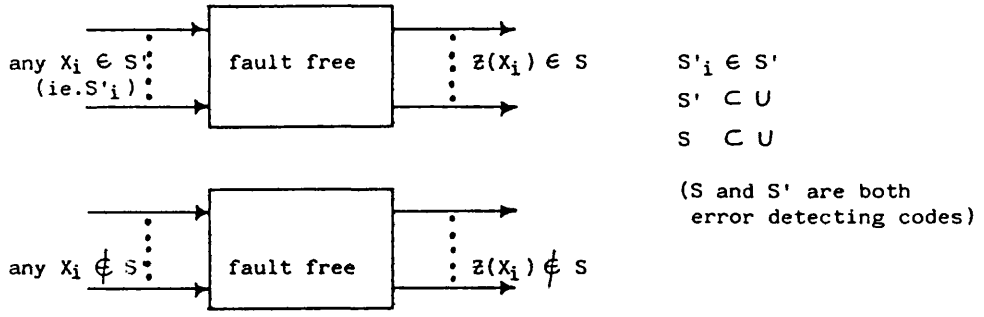


FIGURE 5.7 A CODE DISJOINT CIRCUIT

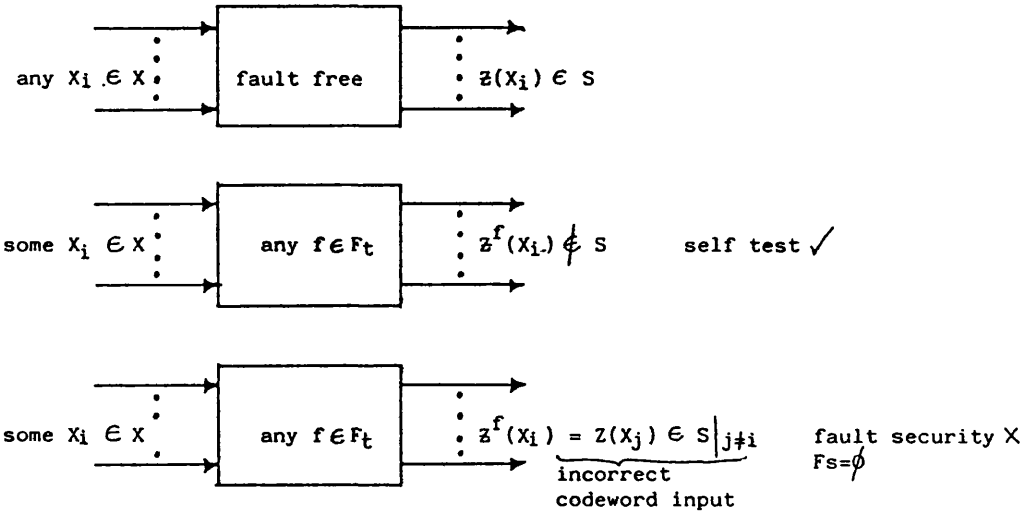


FIGURE 5.8 A SELF TESTING ONLY CIRCUIT

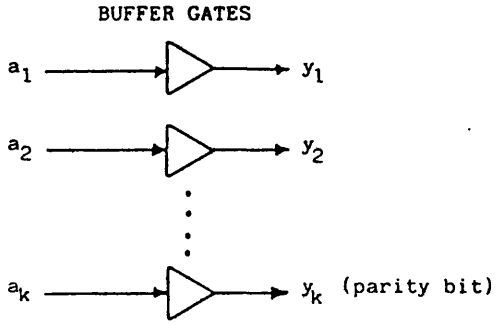


FIGURE 5.9 A TOTALLY SELF CHECKING BUFFER

INPUTS \ FAULT					a_1 SA1						a_1, a_2 SA1						a_1, a_2, a_3 SA1						a_1, a_2, a_3, a_4 SA1					
a_5	a_4	a_3	a_2	a_1	y_5	y_4	y_3	y_2	y_1	R	y_5	y_4	y_3	y_2	y_1	R	y_5	y_4	y_3	y_2	y_1	R	y_5	y_4	y_3	y_2	y_1	R
1	0	0	0	0	1	0	0	0	1	E	1	0	0	1	1	X	1	0	1	1	1	E	1	1	1	1	1	X
0	0	0	0	1	0	0	0	0	1	*	0	0	0	1	1	E	0	0	1	1	1	X	0	1	1	1	1	E
0	0	0	1	0	0	0	0	1	1	E	0	0	0	1	1	E	0	0	1	1	1	X	0	1	1	1	1	E
1	0	0	1	1	1	0	0	1	1	*	1	0	0	1	1	*	1	0	1	1	1	E	1	1	1	1	1	X
0	0	1	0	0	0	0	1	0	1	E	0	0	1	1	1	X	0	0	1	1	1	X	0	1	1	1	1	E
1	0	1	0	1	1	0	1	0	1	*	1	0	1	1	1	E	1	0	1	1	1	E	1	1	1	1	1	X
1	0	1	1	0	1	0	1	1	1	E	1	0	1	1	1	E	1	0	1	1	1	E	1	1	1	1	1	X
0	0	1	1	1	0	0	1	1	1	*	0	0	1	1	1	*	0	0	1	1	1	*	0	1	1	1	1	E
0	1	0	0	0	0	1	0	0	1	E	0	1	0	1	1	X	0	1	1	1	1	E	0	1	1	1	1	E
1	1	0	0	1	1	1	0	0	1	*	1	1	0	1	1	E	1	1	1	1	1	X	1	1	1	1	1	X
1	1	0	1	0	1	1	0	1	1	E	1	1	0	1	1	E	1	1	1	1	1	X	1	1	1	1	1	X
0	1	0	1	1	0	1	0	1	1	*	0	1	0	1	1	*	0	1	1	1	1	E	0	1	1	1	1	E
1	1	1	0	0	1	1	1	0	1	E	1	1	1	1	1	X	1	1	1	1	1	X	1	1	1	1	1	X
0	1	1	0	1	0	1	1	0	1	*	0	1	1	1	1	E	0	1	1	1	1	E	0	1	1	1	1	E
0	1	1	1	0	0	1	1	1	1	E	0	1	1	1	1	E	0	1	1	1	1	E	0	1	1	1	1	X
1	1	1	1	1	1	1	1	1	1	*	1	1	1	1	1	*	1	1	1	1	1	*	1	1	1	1	1	*

ODD PARITY CODEWORDS
a_5=PARITY BIT

Notes : 1) n = 4, 1 parity bit → k = 5
2) R = result status:
a) * = correct codeword output.
b) E = non codeword output (even parity codeword).
c) X = incorrect codeword.
3) Correct outputs = inputs.
4) Input fault ≡ corresponding output with same fault.
5) a_1, a_2, a_3, a_4, a_5 SA1 → $\langle y_5 y_4 y_3 y_2 y_1 \rangle$ permanently 11111.

FIGURE 5.10 A TSC BUFFER WITH FAULTS

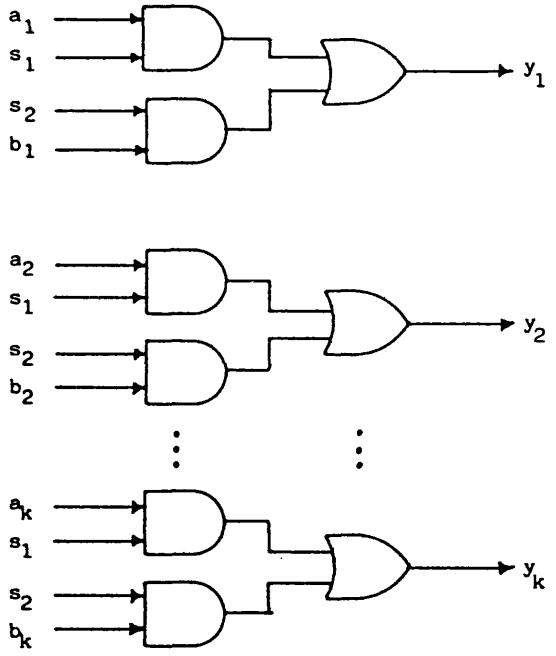


FIGURE 5.11 A TOTALLY SELF CHECKING MULTIPLEXER

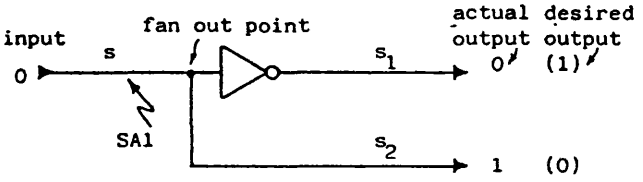
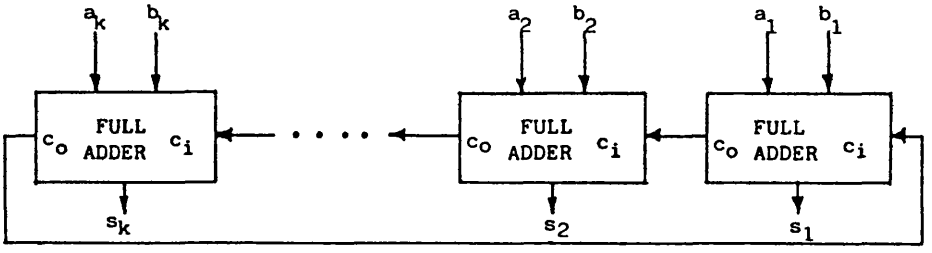
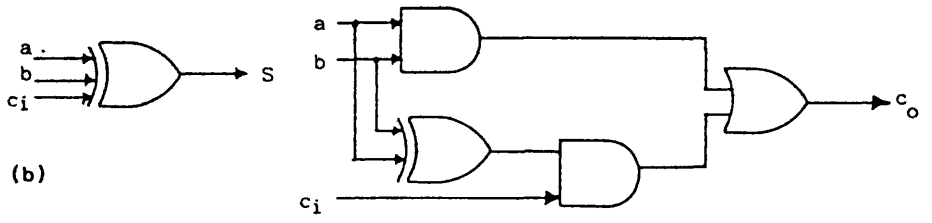


FIGURE 5.12 A CONTROL LINE FAULT



(a)

$s = a \oplus b \oplus c_i$ $\quad c_o = ab + c_i(a \oplus b)$



(b)

FIGURE 5.13 A TOTALLY SELF CHECKING RIPPLE ADDER

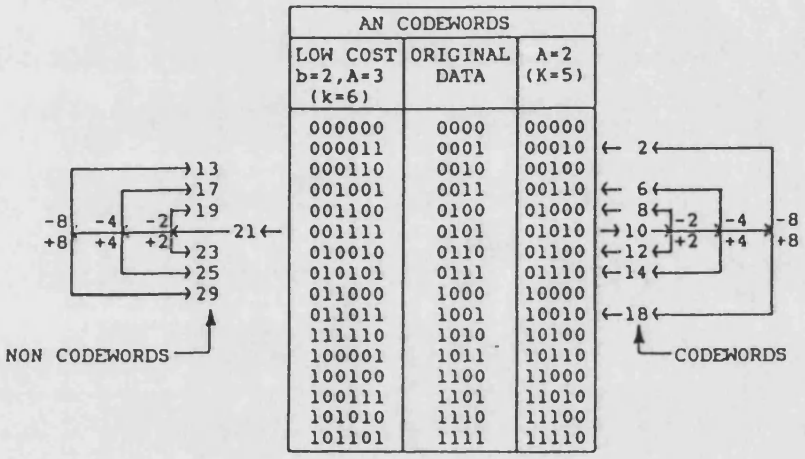


FIGURE 5.14 ERRORS IN AN CODEWORDS ($\pm 2^N$)

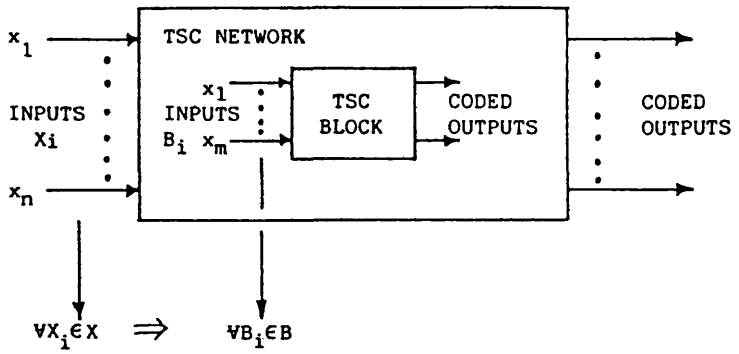


FIGURE 5.15 A FULLY EXCERCISED BLOCK

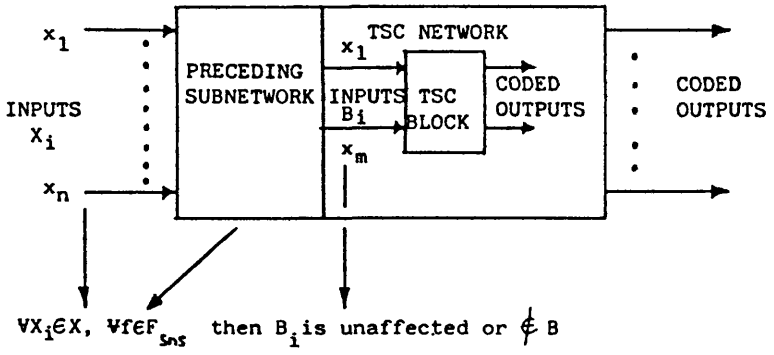


FIGURE 5.16 A SECURELY LOCATED BLOCK

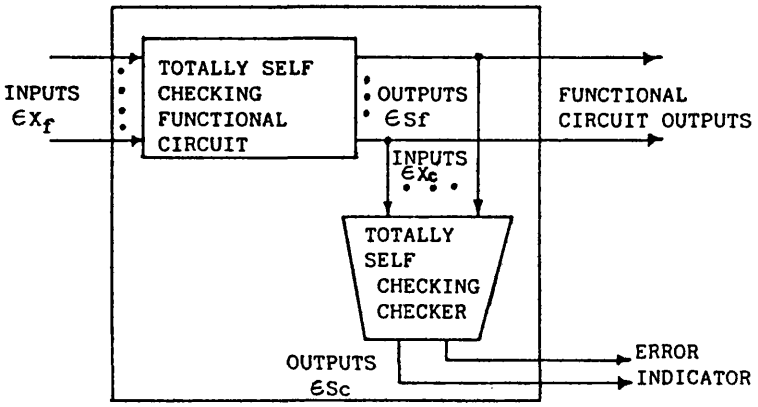


FIGURE 5.17 A TOTALLY SELF CHECKING NETWORK

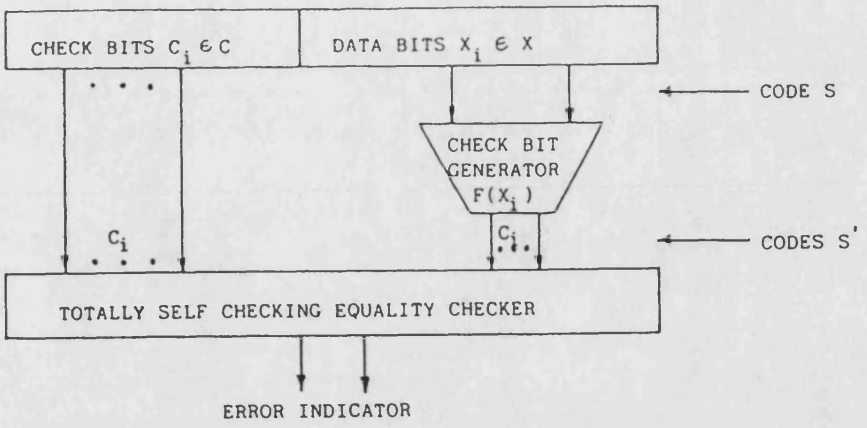


FIGURE 5.18 A TOTALLY SELF CHECKER FOR SEPARABLE CODES

LOW COST CODE
b = 3
A = 7
n = 4
k = 7

DATA BITS X_i	CHECK BITS C_i
0000	000
0001	001
0010	010
0011	011
0100	100
0101	101
0110	110
0111	000
1000	001
1001	010
1010	011
1011	100
1100	101
1101	110
1110	000
1111	001

$X = (0,1)^4$ $111 \notin C$

A = 5
n = 4
k = 7

DATA BITS X_i	CHECK BITS C_i
0000	000
0001	001
0010	010
0011	011
0100	100
0101	000
0110	001
0111	010
1000	011
1001	100
1010	000
1011	001
1100	010
1101	011
1110	100
1111	000

$X = (0,1)^4$ 101, 110, 111 $\notin C$

FIGURE 5.19 RESIDUE CODEWORDS

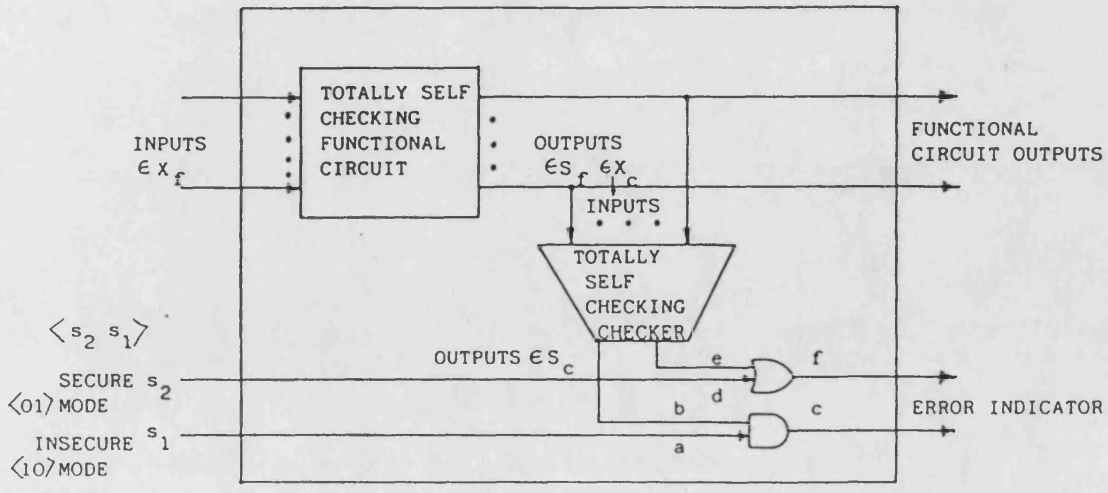


FIGURE 5.20 A TYPE 1 PARTIALLY SELF CHECKING NETWORK

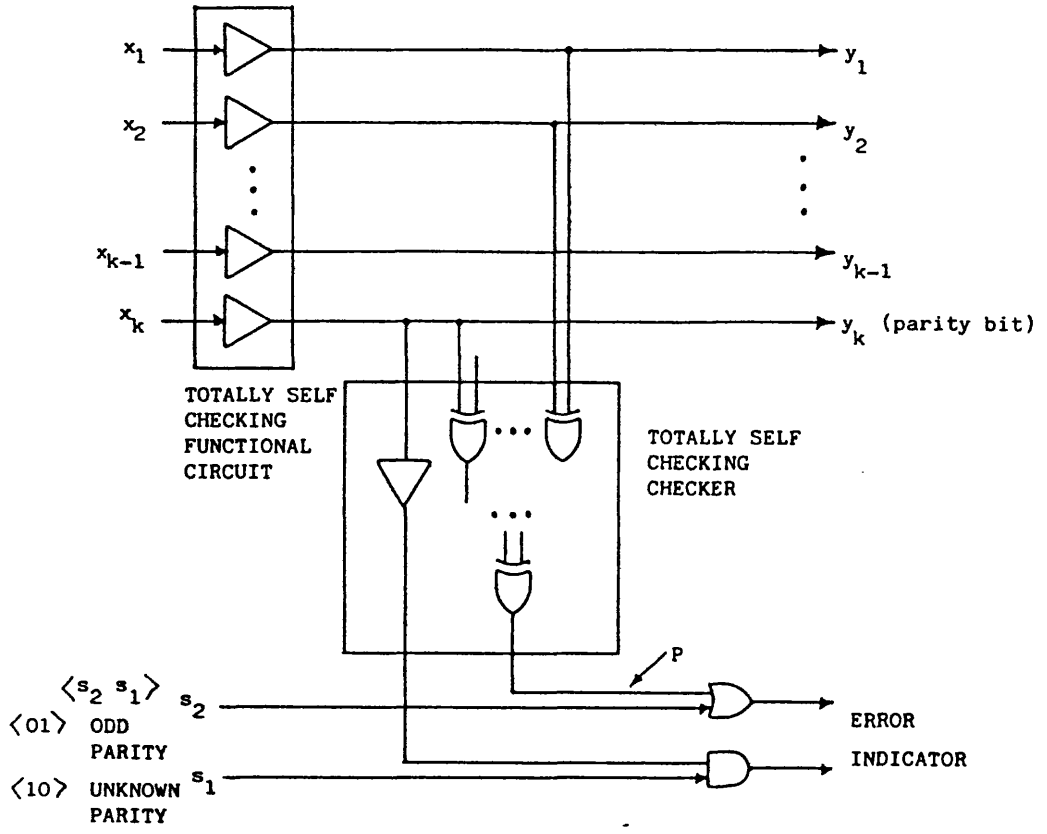


FIGURE 5.21 A PARTIALLY SELF CHECKING PARTY CHECKED BUFFER

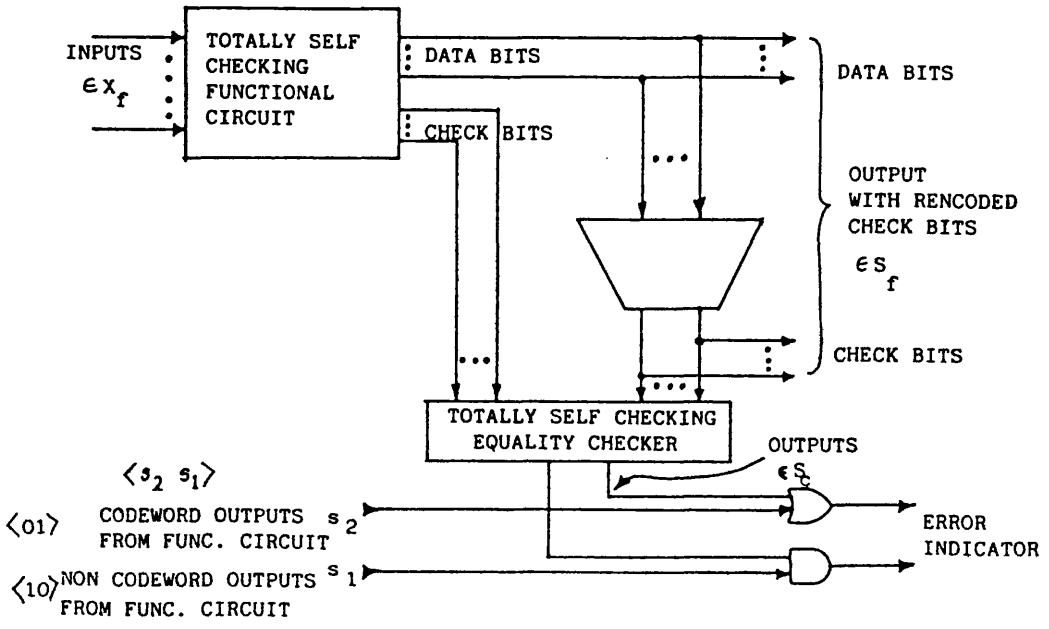


FIGURE 5.22 A TYPE 2 PARTIALLY SELF CHECKING NETWORK

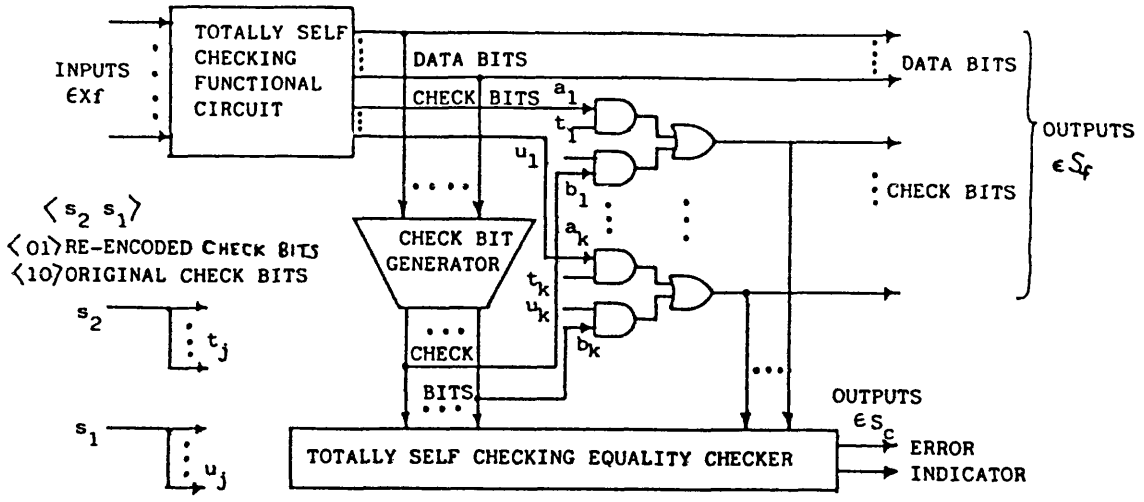
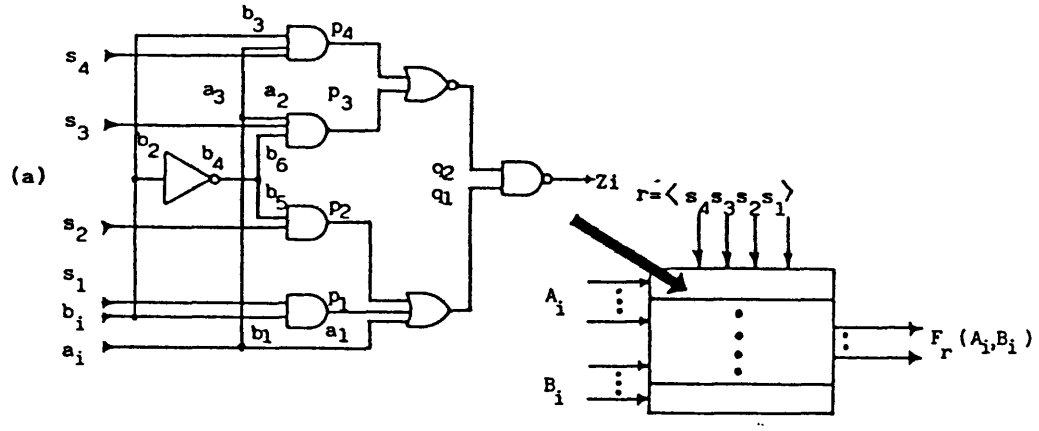


FIGURE 5.23 A TYPE 3 PARTIALLY SELF CHECKING NETWORK



(b)

r=⟨$s_4s_3s_2s_1$⟩	$F_r(A_i,B_i)$	r =⟨$s_4s_3s_2s_1$⟩	$F_r(A_i,B_i)$
0 0 0 0	$\overline{A_i}$	1 0 0 0	$\overline{A_i} + B_i$
0 0 0 1	$\overline{A_i + B_i}$	1 0 0 1	$\overline{A_i \oplus B_i}$
0 0 1 0	$\overline{A_i} \cdot B_i$	1 0 1 0	B_i
0 0 1 1	0	1 0 1 1	$A_i \cdot B_i$
0 1 0 0	$\overline{A_i \cdot B_i}$	1 1 0 0	1
0 1 0 1	$\overline{B_i}$	1 1 0 1	$A_i + \overline{B_i}$
0 1 1 0	$A_i \oplus B_i$	1 1 1 0	$A_i + B_i$
0 1 1 1	$A_i \cdot \overline{B_i}$	1 1 1 1	A_i

FIGURE 5.24 A BIT-SLICE TO PERFORM 16 LOGIC FUNCTIONS ON TWO VARIABLES

FUNCTION $F_r(A_iB_i)$	NODES STUCK-AT-0												
	a_i	a_1	b_i	b_4	s_0	s_1	s_2	s_3	P_1	P_2	P_3	P_4	z_i
0					X	X			X	X			
1							X	X			X	X	X
$A_i \oplus B_i$	X	X	X	X		X	X			X	X		X
$\overline{A_i \oplus B_i}$	X	X	X		X			X	X			X	X
A_i	X				X	X	X	X	X	X	X	X	X
B_i			X	X		X		X		X		X	X
$\overline{A_i}$	X	X											
$\overline{B_i}$			X	X	X		X		X		X		X

FUNCTION $F_r(A_iB_i)$	NODES STUCK-AT-1															
	a_i	a_2	a_3	b_1	b_1	b_3	b_4	b_5	b_6	s_0	s_1	s_2	s_3	q_1	q_2	z_i
0												X	X			X
1										X	X			X	X	
$A_i \oplus B_i$	X	X		X			X	X	X	X			X	X	X	X
$\overline{A_i \oplus B_i}$	X		X	X	X	X					X	X		X	X	X
A_i	X	X	X													X
B_i				X		X	X	X		X		X		X	X	X
$\overline{A_i}$	X									X	X	X	X	X		X
$\overline{B_i}$				X	X		X		X		X		X	X	X	X

Notes : 1) X indicates a detected failure.
2) Only one member from each class of structurally equivalent faults is included.

FIGURE 5.25 FAULT TESTED BY A SUBSET OF FUNCTIONS IN THE CIRCUIT OF FIGURE 5.24a

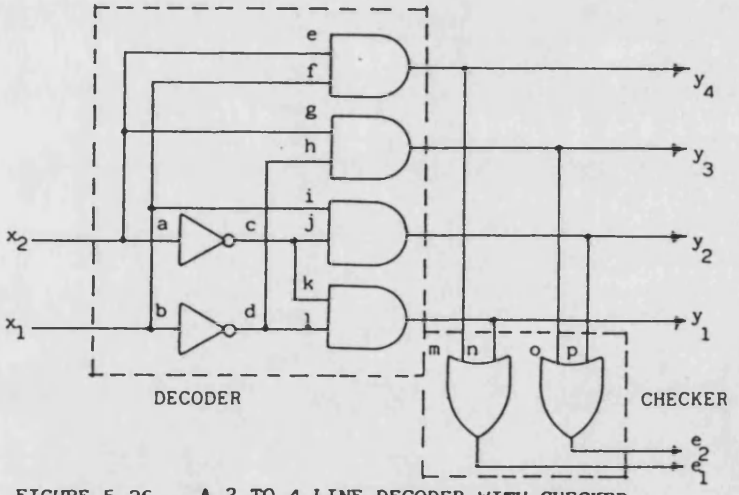


FIGURE 5.26 A 2 TO 4 LINE DECODER WITH CHECKER

NORMAL (FAULT FREE) OPERATION

X_i		ef	gh	ij	kl	Y_i				E_i	
x_2	x_1					y_4	y_3	y_2	y_1	e_2	e_1
0	0	00	01	01	11	0	0	0	1	0	1
0	1	01	00	11	10	0	0	1	0	1	0
1	0	10	11	00	01	0	1	0	0	1	0
1	1	11	10	10	00	1	0	0	0	0	1

KEY

----	--	DECODER AND CHECKER OUTPUTS AS NORMAL
$y_4y_3y_2y_1$	e_2e_1	ERROR IN DECODER OUTPUTS DETECTED BY CHECKER
----	e_2e_1	ERROR IN CHECKER OUTPUTS
$y_4y_3y_2y_1$	--	ERROR IN DECODER OUTPUTS NOT DETECTED BY CHECKER

OPERATION WITH SINGLE STUCK-AT FAULTS

x_2	x_1	a				b				c				d			
		SA0		SA1		SA0		SA1		SA0		SA1		SA0		SA1	
0	0	--	--	0000	00	--	--	0000	00	0000	00	--	--	0000	00	--	--
0	1	--	--	0000	00	0011	11	--	--	0000	00	--	--	--	--	0011	11
1	0	0101	11	--	--	--	--	0000	00	--	--	0101	11	0000	00	--	--
1	1	1010	11	--	--	1100	11	--	--	--	--	1010	11	--	--	1100	11

x_2	x_1	e				f				g				h			
		SA0		SA1		SA0		SA1		SA0		SA1		SA0		SA1	
0	0	--	--	--	--	--	--	--	--	--	--	0101	11	--	--	--	--
0	1	--	--	1010	11	--	--	--	--	--	--	--	--	--	--	--	--
1	0	--	--	--	--	--	--	1100	11	0000	00	--	--	0000	00	--	--
1	1	0000	00	--	--	0000	00	--	--	--	--	--	--	--	--	1100	11

x_2	x_1	i				j				k				l			
		SA0		SA1		SA0		SA1		SA0		SA1		SA0		SA1	
0	0	--	--	0011	11	--	--	--	--	0000	00	--	--	0000	00	--	--
0	1	0000	00	--	--	0000	00	--	--	--	--	--	--	--	--	0011	11
1	0	--	--	--	--	--	--	--	--	--	--	0101	11	--	--	--	--
1	1	--	--	--	--	--	--	1010	11	--	--	--	--	--	--	--	--

x_2	x_1	y_4				y_3				y_2				y_1			
		SA0		SA1		SA0		SA1		SA0		SA1		SA0		SA1	
0	0	--	--	1001	--	--	--	0101	11	--	--	0011	11	0000	00	--	--
0	1	--	--	1010	11	--	--	0110	--	0000	00	--	--	--	--	0011	11
1	0	--	--	1100	11	0000	00	--	--	--	--	0110	--	--	--	0101	11
1	1	0000	00	--	--	--	--	1100	11	--	--	1010	11	--	--	1001	--

x_2	x_1	m				n				o				p			
		SA0		SA1		SA0		SA1		SA0		SA1		SA0		SA1	
0	0	--	--	--	--	--	00	--	--	--	--	--	11	--	--	--	11
0	1	--	--	--	11	--	--	--	11	--	--	--	--	--	00	--	--
1	0	--	--	--	11	--	--	--	11	--	00	--	--	--	--	--	--
1	1	--	00	--	--	--	--	--	--	--	--	--	11	--	--	--	11

x_2	x_1	e_2				e_1			
		SA0		SA1		SA0		SA1	
0	0	--	--	--	11	--	00	--	--
0	1	--	00	--	--	--	--	--	11
1	0	--	00	--	--	--	--	--	11
1	1	--	--	--	11	--	00	--	--

FIGURE 5.27 FAULT TABLE FOR THE CIRCUIT OF FIGURE 5.26

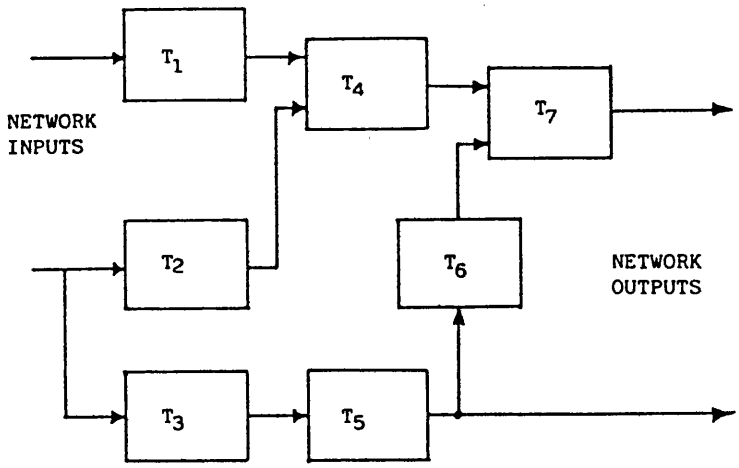


FIGURE 5.28 A SELF TESTING ONLY NETWORK

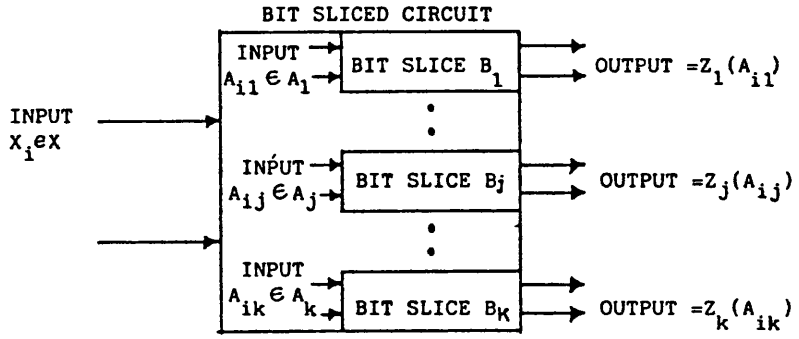


FIGURE 5.29 DEFINITIONS FOR A BIT - SLICED CIRCUIT

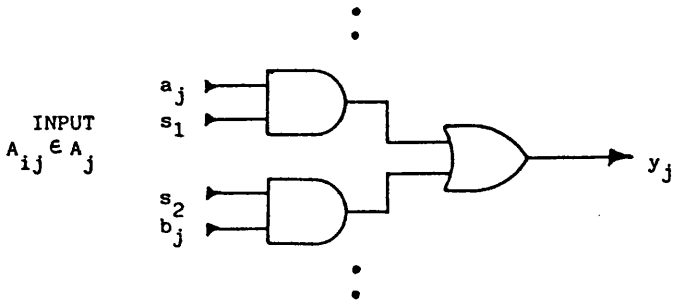


FIGURE 5.30 A BIT-SLICE FROM THE MULTIPLEXER OF FIG. 5.11

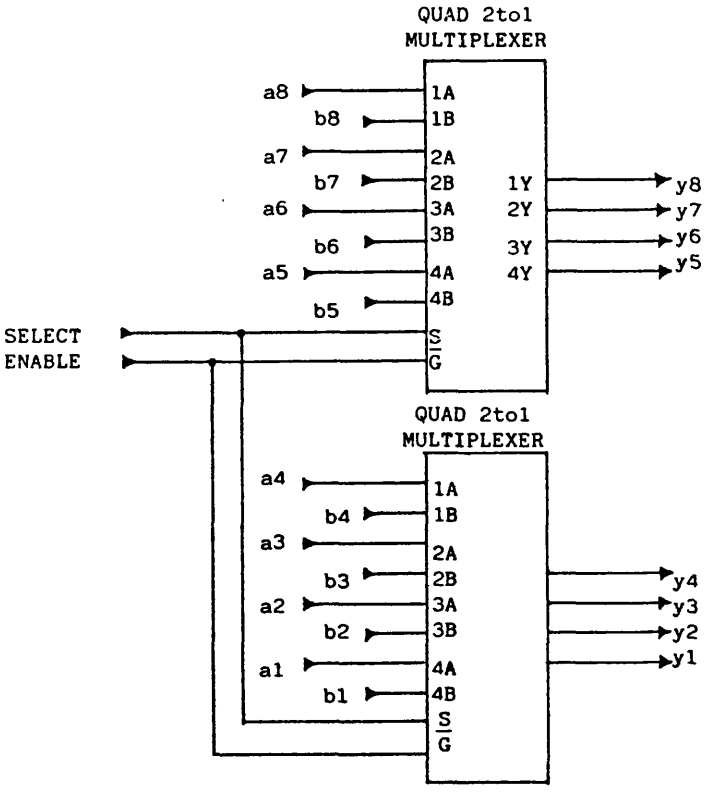


FIGURE 5.31 A BYTE-SLICED MULTIPLEXER

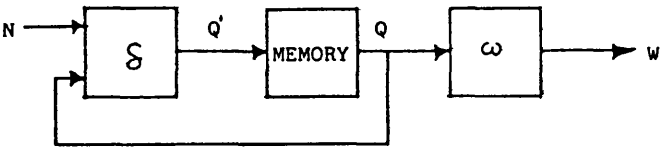


FIGURE 5.32 A SEQUENTIAL MACHINE MODEL

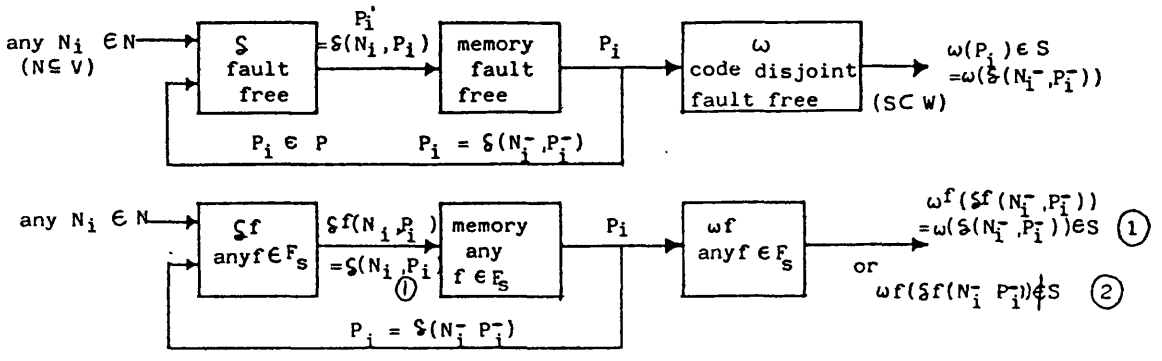


FIGURE 5.33 A FAULT SECURE SEQUENTIAL CIRCUIT

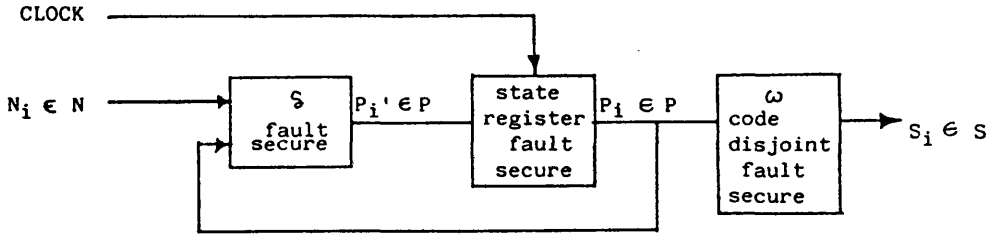
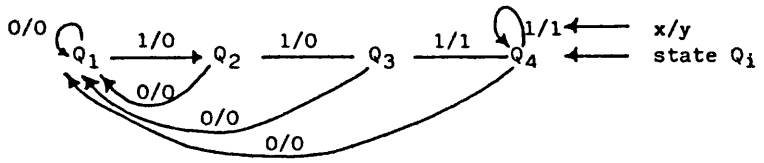


FIGURE 5.34 A FAULT SECURE SYNCHRONOUS SEQUENTIAL MACHINE

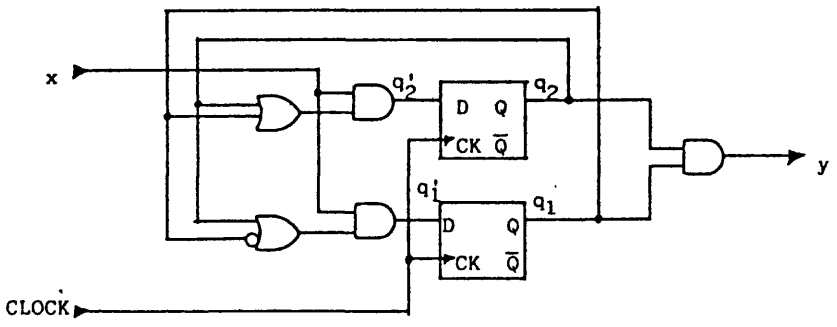


(a) State Diagram

PRESENT STATE $q_2\ q_1$	INPUT x: NEXT STATE 0 $q_2'\ q_1'$	INPUT x: NEXT STATE 1 $q_2'\ q_1'$	OUTPUT y: 0	OUTPUT y: 1
Q_1 0 0	0 0	0 1	0	0
Q_2 0 1	0 0	1 0	0	0
Q_3 1 0	0 0	1 1	0	0
Q_4 1 1	0 0	1 1	1	1

$y = q_2\ q_1$
$q_2' = x(q_2 + q_1)$
$q_1' = x(q_2 + \overline{q_1})$

(b) State Table



(c) Logic Diagram

FIGURE 5.35 SEQUENTIAL MACHINE DESIGN

(a) State Table

P = (even parity codewords)

p = parity bit

S = (⟨01⟩,⟨10⟩)

PRESENT STATE $p_3\ p_2\ p_1$	INPUT x = 0: NEXT STATE $p'_3\ p'_2\ p'_1$	INPUT x = 1: NEXT STATE $p'_3\ p'_2\ p'_1$	INPUT x = 0: OUTPUTS $y_2\ y_1$	INPUT x = 1: OUTPUTS $y_2\ y_1$
P_1 0 0 0	0 0 0	1 0 1	1 0	1 0
P_2 1 0 1	0 0 0 *	1 1 0	1 0	1 0
P_3 1 1 0	0 0 0	0 1 1	1 0	1 0
P_4 0 1 1	0 0 0	0 1 1 *	0 1	0 1

* see section 5.8.2

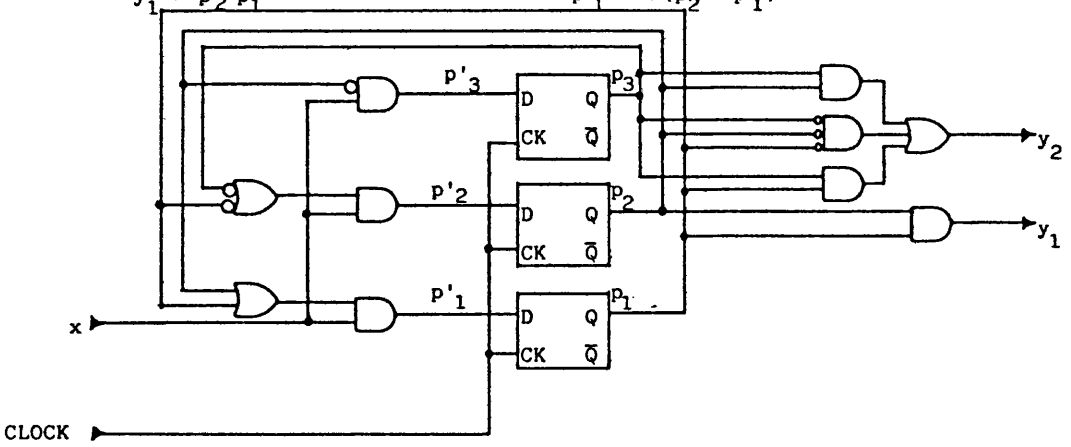
$y_2 = p_3\,p_2 + p_3\,p_1 + \overline{p_3}\,\overline{p_2}\,\overline{p_1}$

$y_1 = p_2\,p_1$

$p'_3 = \overline{x p_2}$

$p'_2 = x(\overline{p_3} + \overline{p_1})$

$p'_1 = x(p_2 + p_1)$



(b) Logic Diagram

FIGURE 5.36 FAULT SECURE DESIGN OF FIG. 5.35

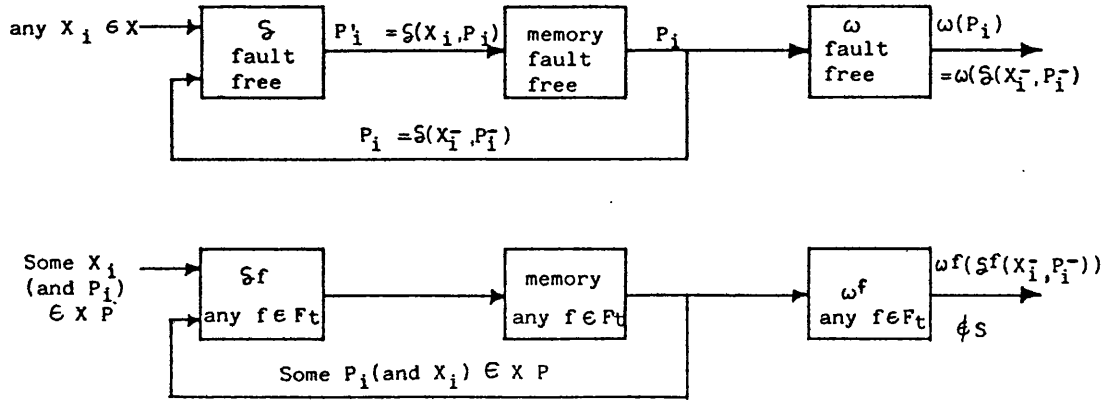


FIGURE 5.37 A SELF TESTING SEQUENTIAL CIRCUIT

CHAPTER SIX : THEORY AND DESIGN OF SELF CHECKING CIRCUITS

6.1 : INTRODUCTION

Chapter 5 has defined the various types of self checking circuit and discussed their properties. This Chapter develops that theory in order to design a number of totally self checking (TSC) circuits.

There is still no universally adopted technique for designing self checking circuits, so it is often necessary to form a self checking network out of a number of smaller blocks, whose self checking abilities are easily demonstrated. The minimal attention paid by researchers in this field to developing any generalised theory has not been helped by many of the papers referenced so far, which present designs, yet give insubstantial explanations of any fundamental theory, their design procedures and how to readily demonstrate that the designs are in fact self checking. This Chapter attempts to remedy that situation with a detailed development of self checking analysis and design from the fault testing requirements of logic gates.

Most research has been devoted to self checking checkers (see section 5.4), a circuit which effectively converts a specific input code to a 1-out-of-2 output code, see Fig. 6.1a. There is also a requirement for self checking circuits with more than one encoded input, illustrated in Fig 6.1b, and self checking circuits with uncoded inputs, illustrated in Fig. 6.1c. Throughout the remaining Chapters only self checking circuits which output a 1-out-of-2 code are considered, for reasons given in section 5.4.1.

It is shown in section 6.5 that a circuit with a 1-out-of-2 code output is inherently fault secure if each output is computed with an independent subcircuit. On this basis only the self testing capabilities of the circuit need to be evaluated for self checking purposes. A number of

theorems are proposed in sections 6.2 to 6.4 which specify the tests required to detect all single stuck-at faults in single and cascaded AND, OR, EXOR and inverter gates. Section 6.4 also introduces the concept of merged test sets, which allow simultaneous testing of more than one gate. The self testing property requires that all the necessary tests are generated within a circuit at some point during normal operation from the application of codeword inputs only. This aspect is discussed in section 6.5.

Section 6.5 applies the theory of section 6.4 to design totally self checking checkers using Karnaugh maps [6.1]. Two theorems specify the conditions for a 2-level AND/OR or OR/AND structure to be self testing and two corollaries show how a Karnaugh map representation of the circuit is used to determine if these conditions are met. Section 6.6 implements this design procedure for an n-input comparator (equality checker), where the input pairs are initially 1-out-of-2 encoded. This circuit appears (mainly for n=2) in virtually any discussion of TSC circuits, but its design has never been adequately documented. An often quoted formula for this design is also derived. Section 6.7 again uses the theory of section 6.4 to design a TSC parity code checker.

The design of a TSC 1-out-of-n code checker is discussed in section 6.8, with specific reference to a 2 to 4 line decoder and its 1-out-of-5 checker. Finally, section 6.9 describes a TSC periodic signal checker. All of these designs are subsequently employed in the self checking computer of Chapter 9.

6.2 : TESTING FOR FAILURES IN AND, OR AND INVERTER GATES

Consider a 2-input AND gate which has four input combinations and six single stuck-at faults. Fig. 6.2 details the faults detected by each input combination. A failure is detected by an incorrect output, so the output stuck-

at-0 (SA0) is detected by an input combination which normally produces a 1 output (<11>) and the output stuck-at-1 (SA1) is detected by an input combination which normally produces a 0 output (<00>, <01> or <10>). An input SA0 is detected by combinations which apply a 1 to that input and levels to the other inputs such that the input failure produces an incorrect output. Thus <11> detects inputs A or B SA0 in a 2-input AND gate (see Fig. 6.2). Similarly an input SA1 is detected by combinations which apply a 0 to that input and produce an incorrect output due to the failure. These are <01> for A SA1 and <10> for B SA1 in a 2-input AND gate. A stuck-at input will not be detected by applying the stuck-at level to it and a stuck-at output will not be detected by an input combination which produces the stuck-at level as an output. Fig. 6.3 details the input combinations to a 2-input OR gate and the single stuck-at faults they detect.

Using this information, it is desirable to ascertain the minimum test set T for each gate. However, this is more clearly observed from the fault tables given in Fig. 6.4 for 3-input AND and OR gates. From this table two theorems are proposed.

THEOREM 6.1 : Every single stuck-at fault in an n-input AND gate is detected with the application of a single 0 to every input (n combinations) and all inputs at 1, a total of n+1 input combinations.

Proof : The only test for an input SA1 is to apply a 0 to that input and a 1 to all others. For n inputs there must be n such tests, a single 0 applied to each input. All these tests produce a 0 at the gate output during fault free operation and a 1 when an input SA1 is detected. They will therefore also detect an output SA1 failure. The only test for an input SA0 is all inputs at 1. Any input SA0 will be detected by this test, which produces a 1 output during fault free operation and a 0 output with a single (or multiple) input SA0 failure. The test there-

fore also detects an output SA0 failure. Thus all single stuck-faults are detected by these n+1 tests. This set of tests is defined as T&, so that T = T&.

THEOREM 6.2 : Every single stuck-at fault in an n-input OR gate is detected with the application of a single 1 to every input (n combinations) and all inputs at 0, a total of n+1 input combinations.

Proof : Using Fig. 6.4. the proof for this theorem is similar to that for theorem 6.1. This set of tests is defined as T+, so that T = T+.

Although self evident, the testing requirments for an inverter gate are given for completeness by the following theorem.

THEOREM 6.3 : Every single stuck-at fault in an inverter gate is detected with application of a 0 and 1 to its input.

Proof : Fig. 6.5 gives the fault table for an inverter gate. This shows that a 0 input will detect input SA1 and output SA0 failures, whereas a 1 input will detect input SA0 and output SA1 failures; i.e. both input combinations are required to fully test the gate. This set of tests is defined as T', so that T = T'.

6.3 : TESING FOR FAILURES IN EXCLUSIVE-OR GATES

The procedures for testing stuck-at failures in an Exclusive-OR (EXOR) gate are somewhat different to those in section 6.2 and hence are dealt with separately. Fig. 6.6 gives the fault tables for 2, 3 and 4-input EXOR gates. From these it can be seen that there are 2^{n-1} (n is the number of inputs) combinations which will test for every single stuck-at failure. Half of the inputs which detect an input stuck-at failure produce a 1 output from the gate during fault free operation and the other half

produce a 0 output. It is therefore much more difficult to derive a minimum test set. Any input combination and its inverse will detect all input SA faults in all gates, for example ‹000› and ‹111› when n = 3. These two tests will also detect both output failures in gates with odd n (n=3 in Fig. 6.6). They will not do the same for gates with even n because both produce the same output (n=2 and n=4 in Fig. 6.6). In this case a third test is necessary which produces an output opposite to that of the first two. From this discussion the following theorem is proposed.

THEOREM 6.4 : Every single stuck-at failure in an n-input EXOR gate is detected with the application of input combinations which provide a 0 and a 1 to every input. There must also be at least one input combination which has an odd number of 1's and at least one input combination which has an even number of 1's.

Proof : Single input SA0 failures are detected by applying a 0 to every input and single input SA1 failures are detected by applying a 1 to every input. The combinations are therefore not as restrictive as those for AND and OR gates. They can range from a single 0 and a single 1 applied to all inputs to the minimum set of two tests (any combination and its inverse). The only restriction is that all inputs receive a 0 and a 1. This is possible since, given any input combination, a single change in this combination due to a fault will produce a change in the output and will thus be detected. The requirement for at least one input combination which has an odd number of 1's and at least one input combination which has an even number of 1's ensures that the output of the gate will be a 1 and 0 respectively, so that both output faults can be detected. This set of tests is defined as $T_{\oplus}$.

6.4 : TESTING FOR FAILURES IN CASCADED GATES

So far, the tests for single stuck-at faults have been for

gates in isolation. In practice gates will be cascaded, so the effect this has on testing them for failures must be considered. The requirements of theorems 6.1-6.4 must still be met for each gate, to ensure that all failures within them can be detected. Assuming that this is the case, a detected failure within the circuit needs to be propagated to its output(s) so it can be detected externally by an error indicator or checker. Theorem 6.5 gives the conditions for this to occur.

THEOREM 6.5 : Every test $t \in T$ applied to a single gate, where T is the test set for that gate (and given by theorems 6.1-6.4), has an output y^f, the level it occupies with the fault present, which is the inverse of its normal output. All single stuck-at faults in the circuit consisting of a gate whose output y is connected to an input x_i of a second gate can be detected at the output of the second gate, if $\forall\ t \in T_1$ are applied to the first gate, where T_1 is T for that gate, and for at least one occurrence of each t there must be an input combination to the second gate which tests for input x_i stuck-at-y^f_t, where y^f_t is y^f for test t. In addition $\forall\ t \in T_2$ must be applied to the second gate, where T_2 is T for that gate.

Proof : Consider a 3-input AND gate feeding a 3-input OR gate, as shown in Fig. 6.7a. A fault in the AND gate needs to be propagated to the output of the OR gate. With the application of $\forall\ t \in T_1 = T\&$, a fault in the AND gate is detected when its output is opposite to that expected. It is this faulty level, d^f, which must be propagated to output h. There is a d^f for each t, i.e. d^f_t, so that for at least one occurrence of each t (t could occur more than once) there must be an input combination to the OR gate which detects f stuck-at-d^f_t. For example, in Fig. 6.7a, a = b = c = 1 tests for a, b, c or d SA0. During fault free operation d=1, but for any of these faults d=0. To propagate this fault the OR gate must have the appropriate input combination to detect f SA0, i.e. a single 1 on f, ⟨efg⟩ = ⟨010⟩. From Fig. 6.7b the four conditions to

detect and propagate all faults in the AND gate require ⟨efg⟩ = ⟨000⟩ and ⟨010⟩. However the OR gate requires $\forall t \in T_2 = T+$ to fully test it, so ⟨efg⟩ = ⟨100⟩ and ⟨001⟩ must also occur.

Example 6.1 : The structure of Fig. 6.7a is expanded in Fig. 6.8a to give a complete two level AND/OR structure. There are now three 3-input AND gates to test, so the procedure adopted in Fig. 6.7b to test for a fault and propagate it is detailed for each AND gate in turn.

The fault propagation requirement means that whilst one AND gate is being tested, the other two must have any input combination except ⟨111⟩ so that they output a 0, denoted by X in Fig. 6.8a. As a result, the required OR gate input combinations automatically satisfy the conditions to fully test this gate, see Fig. 6.8b. From Fig. 6.8b there are twelve conditions which must be satisfied to fully test the circuit for all single stuck-at faults. However, this can be reduced to a minimum of six tests as follows. The tests for a SA1 fault in any AND gate (inputs ⟨011⟩, ⟨101⟩ and ⟨110⟩) all require the OR gate input to be ⟨000⟩. These tests produce a 0 output from the tested gate during fault free operation, so they can be merged without affecting the fault propagation requirements. This allows all AND gates to be simultaneously tested for SA1 faults. Any single SA1 fault in any of the three AND gates will then produce a 1 output from that gate and a change in the OR output from 0 to 1. The tests can be merged in any combination. Fig. 6.8c gives two examples. The tests for SA0 failures in the AND gates cannot be merged in a similar manner, since the propagation requirement allows only the tested AND gate output to be a 1. SA0 tests cannot be merged with a SA1 test either, as a detected fault in the former would not be propagated, since the OR gate already has an input at 1 due to the latter. (Another due to a SA0 fault will not cause a change at its output.) If the SA1 tests are fully merged, the circuit in Fig 6.8a requires a minimum of six

tests, as indicated in Fig. 6.8c.

Fig. 6.9 extends the concepts of Fig 6.8 to give the test requirements for various cascaded OR/AND, AND/AND, OR/OR and mixed structures. These structures are totally self checking for all single stuck-at faults if all the first level gates are fully tested. An n-input AND or OR gate requires n+1 tests to fully test it (from theorems 6.1 and 6.2), so the total number of unmerged tests T_u for the 2-level structures in Fig. 6.9 is given by:

$$T_u = \sum_{i=1}^{m} (n_i + 1) \quad : \quad \text{where } n_i \text{ is the number of inputs for gate i and m is the number of first level gates.} \qquad (6.1)$$

To calculate the minimum number of tests which occur in a fully merged test set, consider 2-level AND/OR and OR/AND structures only, with m x n-input first level gates. The merging process is applied to SA1 tests for the AND/OR structures and SA0 tests for the OR/AND structures. In either case these tests constitute n of the n+1 tests for each gate (see proof of theorem 6.1). In a fully merged test set these n tests for m-1 of the first level gates are combined with those for the remaining gate. For the circuit conditions given above, $T_u = m(n+1)$, so the number of fully merged tests T_m is then given by:

$$T_m = m(n+1) - (m-1)n = m+n \qquad (6.2)$$

Applying equation (6.2) to example 6.1, for which m=3 and n=3, gives T_u=12 and T_m=6, as expected.

Note that for AND/AND and OR/OR structures the only tests which can be merged are the SA0 and SA1 tests respectively, which in fact are automatically merged due to the requirements of the second level gate for fault propagation, see Fig. 6.9b and 6.9c.

A corollary is now presented to theorem 6.5 for the casca-

ding of EXOR gates.

Corollary 6.1 : All single stuck-at failures in a circuit consisting of an EXOR gate whose output y is connected to an input x_i of a second gate can be detected at output of a second EXOR if, $\forall$ t $\in$ T$\oplus$ are applied to both gates.

Proof : Consider a 2-input EXOR gate feeding another 2-input EXOR gate, as shown in Fig. 6.10a. Using theorem 6.4, Fig 6.10b details a set T = T$\oplus$ for the first gate and possible input combinations to the second gate to propagate detected faults. As stated before, a single bit change to an EXOR input combination will always change its output during fault free operation, so if c changes to c^f in the first gate, this level will always be propagated through the second. It is only necessary, therefore, to ensure that $\forall$ t $\in$ T$\oplus$ are applied to each gate.

So far, only two cascaded gates have been considered. If, as is likely, more than two gates are cascaded, all single failures in every gate need to be propagated to the final output(s). To ensure that this happens, theorem 6.5 must now be applied to each pair of connected gates in the circuit and its principles extended until the final output is reached. Using Fig. 6.11a as an example, the procedure is as follows:

1) Start at gate A:
   a) For $\forall$ t $\in$ T& (gate A), determine appropriate t $\in$ T+ for gate B from theorem 6.5.
   b) For the t $\in$ T+ required for gate B in a), determine appropriate t $\in$ T& for gate C from theorem 6.5.
2) Start at gate B:
   a) For $\forall$ t $\in$ T+ (gate B), determine appropriate t $\in$ T& for gate C from theorem 6.5. Some of these tests will have already been provided by 1a).
3) Ensure that $\forall$ t $\in$ T& are applied to gate C. Again,

some of these will have already been provided by 1b) and 2a).

Fig. 6.11b gives the results of 1), 2) and 3) for the circuit in Fig. 6.11a.

6.5 : DESIGN OF TSC CHECKERS USING KARNAUGH MAPS

A self checking checker maps a coded input (or inputs) to a 1-out-of-2 code output, as shown in Fig. 6.1a. Its normal input set is X_c and its output set is S_c, where S_c = {⟨01⟩,⟨10⟩}. From section 5.4.1, two fundamental rules must be applied to every TSC checker design. These are:

1) The checker must be code disjoint:
   a) Code inputs are mapped to outputs ⟨01⟩ or ⟨10⟩.
   b) Non code inputs are mapped to outputs ⟨00⟩ or ⟨11⟩.
2) Both code outputs must be used:
   a) At least one code input must be mapped to output ⟨01⟩.
   b) At least one code input must be mapped to output ⟨10⟩.

A checker must be totally self checking on the basis of codeword inputs only. If an independent subcircuit is used for each output, the checker will be fault secure for all faults affecting one subcircuit only. Under these fault conditions it will output the correct codeword or a non codeword. It is also fault secure for faults creating a unidirectional error at its outputs. In general, only single stuck at faults are considered for analytical and design purposes. Independent subcircuits will be adopted for all designs, so only the self testing capabilites of each subcircuit need to be evaluated.

A design procedure is now presented for TSC checkers using Karnaugh maps [6.1]. From a Karnaugh map any circuit

design can be implemented using either a sum of products (normal minterm form) or a product of sums (normal maxterm form) [6.1]. A sum of products implementation is a two level AND/OR structure as shown in Fig. 6.8a, whilst a product of sums implementation is a two level OR/AND structure as shown in Fig. 6.9a. Every design requires a Karnaugh map for each output. The size of the map is determined by the number of network inputs. Code inputs are depicted by a circle within the appropriate minterm square on each map.

6.5.1 : TSC AND/OR STRUCTURES

The following theorem ensures that an AND/OR structure is self testing for code inputs.

THEOREM 6.6 : A 2-level AND/OR structure is self testing for all single stuck at faults if code inputs provide:

i) At least one occurrence of a single 0 on each input of every AND gate with no other AND gate output at 1.
ii) At least one occurrence of an all 1 input to each AND gate with no other AND gate output at 1.

Proof : The single 0 and all 1 requirements for each gate are the conditions of theorem 6.1 to detect all single stuck-at faults in an AND gate, whilst the occurrence of these with no other AND gate output at 1 satisfies the condition of theorem 6.5 to propagate a fault. Section 6.4 and Fig. 6.8 have already demonstrated that the single 0 tests can be merged and that the requirements of theorem 6.5 for the second level OR gate will be automatically met.

A means of interpreting this theorem on a Karnaugh map is now required. Before presenting a corollary to theorem 6.6 for this purpose, a definition is required.

Definition 6.1 : The expansion of a prime implicant [6.1]

with respect to one of its constituent variables is the area on a Karnaugh map where that variable alone is at 0.

Fig. 6.12 shows these areas on a 4 variable Karnaugh map for each variable of a 2, 3 and 4 variable prime implicant.

Corollary 6.2 : A 2-level AND/OR structure is self testing for all single stuck-at faults from code inputs if the following conditions are satisfied on a Karnaugh map:

i) In the expansion of each prime implicant with respect to each constituent variable, there must be at least one codeword input which results in a 0 output; i.e. a ⓪.
ii) Each prime implicant must contain at least one codeword input which is unique to that prime implicant, i.e. not covered by more than one prime implicant.

Proof : A codeword containing a 0 (⓪) in the expansion of a prime implicant with respect to a variable, results in this variable alone being a 0 for that code input. A codeword containing a 1 (①) in the expansion of a prime implicant with respect to a variable, results in a 1 at the output of the OR gate, i.e. one of the other AND gates has a 1 output for that code input. Whilst this latter condition is allowed, there must be at least one instance where it does not occur (for codeword inputs), so the expansion of the prime implicant must contain at least one ⓪. In order to provide a single 0 to each input of every AND gate, there must be at least one ⓪ in the expansion of every prime implicant with respect to all of its variables.

Every prime implicant must contain a codeword. If it does not, the AND gate it represents may receive all input combinations which have a single 0, but it will never receive the all 1 input combination to test this gate from code inputs. This all 1 input combination also provides a

single 1 test to the second level OR gate. A codeword which is covered by more than one prime implicant results in all these prime implicants (AND gates) presenting a 1 to the second level OR gate for that code input. Whilst this condition is allowed, there must be at least one instance where it does not occur (for code inputs), so that each prime implicant must contain at least one code input which is not covered by any other prime implicant.

Fig 6.13 illustrates these points in a number of proposed designs for one output of a TSC checker, which are not self testing and therefore not in fact TSC for the reasons given in each case.

A prime implicant which contains an inverted variable does not create a problem as Fig. 6.14 indicates. The AND gates still receive all their required test inputs and since every AND gate input has to be 0 for some tests and 1 for others, the added inverter gate will be automatically tested to the conditions given in theorem 6.3.

6.5.2 : TSC OR/AND STRUCTURES

The following theorem ensures that an OR/AND structure is self testing from code inputs.

THEOREM 6.7 : A 2-level OR/AND structure is self testing for all single stuck at faults if code inputs provide:

i) At least one occurrence of a single 1 on each input of every OR gate with no other OR gate output at 0.
ii) At least one occurrence of an all 1 input to each OR gate with no other OR gate output at 0.

The proof is similar to that for theorem 6.6 and is therefore not included. A circuit design can be implemented just as easily in an OR/AND form from a Karnaugh map as the AND/OR form [6.1]. Prime implicants now become prime implicates, covering 0's instead of 1's. Definition 6.1

now needs to be modified.

Definition 6.2 : The expansion of a prime implicate with respect to one of its constituent variables is the area on a Karnaugh map where that variable alone is at 1.

Fig. 6.15 shows these areas on a 4 variable Karnaugh map for each variable of a 2, 3 and 4 variable prime implicate.

A similar corollary to that for theorem 6.6 can now be presented.

Corollary 6.3 : A 2-level OR/AND structure is self testing for all single stuck-at faults from code inputs if the following conditions are satisfied on a Karnaugh map:

i) In the expansion of each prime implicate with respect to each constituent variable, there must be at least one codeword input which results in a 1 output; i.e. a (1).
ii) Each prime implicate must contain at least one codeword input which is unique to that prime implicate; i.e. not covered by more than one prime implicate.

Again the proof is not included, as it is so similar to that for corollary 6.2

TSC circuits and networks can now be designed using the above theorems and corollaries.

6.6 : A TSC N-BIT COMPARATOR

A TSC circuit is frequently required to compare two n-bit words. Consider, initially, that each input and its corresponding input in the other word are encoded in a 1-out-of-2 code, in the manner of Fig. 6.16. This circuit is then an n x 1-out-of-2 to 1 x 1-out-of-2 code converter.

Consider two pairs of inputs $\langle a_1 b_1 \rangle$ and $\langle a_2 b_2 \rangle$, so that there are four possible code inputs $\langle a_1 b_1 a_1 b_1 \rangle$; ⟨0101⟩, ⟨0110⟩, 1001⟩ or ⟨1010⟩. Each of these must be mapped to one of the code outputs ⟨01⟩ or ⟨10⟩, with each output occurring at least once. Non code inputs are mapped to the non code outputs ⟨11⟩ or ⟨00⟩ for the circuit to be code disjoint. In addition, the circuit must be self testing from only the four code inputs.

Circuit options for each output are:

1) A single AND (OR) gate with up to three inputs, since the number of tests for a 3-input gate is four.
2) A single EXOR gate, since a minimum of two or three tests are required for an n-input gate, where n is odd or even respectively.
3) A 2-level AND/OR (or OR/AND) structure wth up to 2 x 2-input AND gates, since the number of tests when fully merged is four.

A single AND with two or three inputs is represented on a Karnaugh map by a block of two or four 1's respectively. Both of these prime implicants can only cover a single codeword, so option 1) is eliminated. Example 6.2 indicates that option 2) can provide a design that is self testing but not code disjoint, whereas example 6.3 shows that option 3) is the most viable.

Example 6.2 : The EXOR/EXNOR combination in Fig. 6.17a provides the desired function. Its truth table in Fig. 6.17b demonstrates that it is self testing for code inputs, but also shows that it is not code disjoint.

Example 6.3 : Option 2) requires a block of four 1's on a 4 variable Karnaugh map for each 2-input AND gate. The design of Figs. 6.18a and 6.18b is a 2-level AND/OR structure which is self testing from corollary 6.2. Its output $\langle y_1 y_2 \rangle$ is given by:

$$y_1 = a_1b_2 + b_1a_2$$
$$\text{and} \quad y_2 = a_1a_1 + b_2b_2 \qquad (6.3)$$

The structure of each subcircuit is identical, so Fig. 6.18c details the testing requirements for one of them. This process has already been demonstrated in example 6.1. It shows that the inputs to each subcircuit must satisfy six conditions to fully test for all single stuck-at faults and that this number may be reduced to four in a fully merged test set. However, there are only four codeword inputs, so each subcircuit MUST have a fully merged test set, with each codeword generating one of the four tests. The truth table of the circuit in Fig. 6.18d demonstrates that it is code disjoint and that it uses a fully merged test set for self testing purposes. Fig. 6.18e gives the fault table for the subcircuit which computes y_1, which confirms that all code inputs are required to test it. It is interesting to note that the second level OR gates provide a 1-out-of-4 to 1-out-of-2 code conversion. This process is not, however, code disjoint. The code disjoint property of the overall circuit is provided by the first level AND gates.

Example 6.4 : In a similar manner to example 6.3, the design of Figs. 6.19a and b is a 2-level OR/AND structure which is self testing from corollary 6.3. Its output $\langle y_1y_2 \rangle$ is given by:

$$y_1 = (a_1 + a_2)(b_1 + b_2)$$
$$\text{and} \quad y_2 = (a_1 + b_2)(b_1 + a_2) \qquad (6.4)$$

Figs. 6.19c and d demonstrate that it is code disjoint and uses a fully merged test set. The second level AND gates in this instance provide a 3-out-of-4 to 1-out-2 code conversion which is not code disjoint.

The circuits of examples 6.3 and 6.4 are those presented by Carter [6.2] and Anderson [6.3]. They are now expanded

for more than two input pairs and henceforth referred to as 2-input Morphic AND gates.

A Morphic AND gate is a conventional AND gate where each input and output is replaced by a pair of inputs and a pair of outputs respectively, with each pair encoded in a 1-out-of-2 code. The output is a codeword if all input pairs are codewords. Thus the output pair $\langle y_1 y_2 \rangle$ of a 2-input Morphic AND gate is given by:

$$\langle y_1 y_2 \rangle = \langle a_1 b_1 \rangle \,\&_M\, \langle a_2 b_2 \rangle \qquad (6.5)$$

where $\&_M$ is defined as the Morphic AND operation. An n-input Morphic AND gate has n input pairs, i.e. 2n physical inputs. Fig. 6.20 compares the truth tables for conventional and Morphic 2-input AND gates. The truth table of the latter confirms that examples 6.3 and 6.4 are 2-input Morphic AND gates.

If y_{n1} and y_{n2} denote the outputs of an n x 1-out-of-2 to 1 x 1-out-of-2 converter, or n-input Morphic AND gate, then for n=1:

$$\begin{aligned} y_{11} &= a_1 \\ y_{12} &= b_1 \end{aligned} \qquad (6.6)$$

Using equations (6.4) and (6.6), the outputs for n=2 are given by:

$$\begin{aligned} y_{21} &= a_1 b_2 + b_1 a_2 = y_{11} b_2 + y_{12} a_2 \\ y_{22} &= a_1 a_2 + b_1 b_2 = y_{11} a_2 + y_{21} b_2 \end{aligned} \qquad (6.7)$$

A 3-input Morphic AND gate, illustrated in Fig. 6.21a, consists of two cascaded 2-input Morphic AND gates [6.4], so that;

$$\begin{aligned} y_{31} &= y_{21} b_3 + y_{22} a_3 \\ &= a_1 b_2 b_3 + b_1 a_2 b_3 + a_1 a_2 a_3 + b_1 b_2 a_3 \\ y_{32} &= y_{21} a_3 + y_{22} b_3 \end{aligned}$$

$$= a_1b_2a_3 + b_1a_2a_3 + a_1a_2b_3 + b_1b_2b_3 \qquad (6.8)$$

Equation (6.8) indicates that the 3-input Morphic AND gate can also be implemented as the 2-level AND/OR structure of Fig. 6.22a. However, it needs to be established if this circuit or that in Fig. 6.21a is totally self checking. Each 2-input Morphic AND gate in Fig. 6.21a is TSC (inherently fault secure and self testing from theorem 6.6), code disjoint, fully exercised (the second level Morphic AND gate also receives all input codewords, see the truth table in Fig. 6.21b) and there is no reconvergent fanout in the network. Thus the conditions of corollary 5.1 are satisfied and the network is TSC. Applying corollary 6.2 to the Karnaugh map representation of equation (6.8) in Fig. 6.22b reveals that the 2-level AND/OR structure is also self testing and thus TSC. The truth table for output y_{31} in Fig. 6.22c also shows that this implementation requires all codeword inputs to fully test it, but does not use a fully merged test set (seven tests).

From Fig. 6.23a a 4-input Morphic AND gate is given by:

$$\begin{aligned} y_{41} &= y_{31}b_4 + y_{32}a_4 \\ y_{42} &= y_{31}a_4 + y_{32}b_4 \end{aligned} \qquad (6.9)$$

Equation (6.9) can again either be implemented as in Fig. 6.23a, or as the 2-level AND/OR structure shown in Fig. 6.23b. Both are TSC.

In general, from equations (6.7) to (6.9), an n-input Morphic AND gate is given by:

$$\begin{aligned} y_{n1} &= y_{(n-1)1}b_n + y_{(n-1)2}a_n \\ y_{n2} &= y_{(n-1)1}a_n + y_{(n-1)2}b_n \end{aligned} \qquad (6.10)$$

Equation (6.10) is that presented by Carter [6.5] and Anderson [6.3].

Equations (6.7) to (6.9) also show that each output has 2^{n-1} prime implicants, each with n variables. This means that both subcircuits require 2^{n-1} x n-input AND gates and a 2^{n-1}-input OR gate. The n-input Morphic AND gate therefore satisfies the conditions associated with equation (6.2), which gives the number of fully merged tests as $T_m = 2^{n-1}+n$. The number of possible codeword inputs to an n-input Morphic AND gate is 2^n, since each input has two code combinations (<01> and <10>). Equating the number of codewords to T_m reveals that only a 2-input Morphic AND gate uses a fully merged test set.

Consider now a 4-input Morphic AND gate formed from 2 x 2-input Morphic AND gates, both implemented as an AND/OR structure, which feed a third 2-input Morphic AND gate implemented as an OR/AND structure, see Fig. 6.24a. Each 2-input Morphic AND gate is TSC, so the use of mixed structures in no way affects the self checking ablility of the network. In Fig. 6.24a, there is a set of 2 x 2-input OR gates feeding a third 2-input OR. These gates can be merged to give a set of 4-input OR gates and the circuit of Fig. 6.24b. The second level of the AND/OR structure has now been merged with the first level of the OR/AND structure, which results in fewer gates and gate levels. This merging process does not affect the self checking ability of the network either, because the minimum test set for the cascaded 2-input OR gates (which must exist in the original network) is identical to that for the single 4-input OR gate of Fig. 6.25. This set of OR gates forms a non code disjoint 2 x 1-out-of-4 to 1 x 3-out-of-4 code converter.

If the 4-input Morphic AND gate is formed from 2 x 2-input OR/AND Morphic AND gates, which feed a 2-input AND/OR Morphic AND gate, then a similar merging process can be performed on the central AND gates, as shown in Fig. 6.26. This set of AND gates form a non code disjoint 2 x 3-out-of-4 to 1 x 1-out-of-4 code converter.

A Morphic AND gate for a large number of inputs can use both of these merging processes. This significantly reduces the number of gates and gate levels when compared to the cascading of original 2-input Morphic AND gates. Fig. 6.27 shows both of these networks for n=8. The third option, a 2-level AND/OR structure, would require a total of 256 x 8-input AND gates and 2 x 128-input OR gates.

All the circuits in this section so far have dealt with input pairs encoded in a 1-out-of-2 code. The original intention was to design a TSC n-bit comparator or equality checker. In this case the input pair is effectively encoded in the duplication code, i.e. both inputs identical, as opposed to the 1-out-of-2 code in which they are always opposite. It is therefore a simple matter to convert all of the above designs into equality checkers by inserting an inverter gate into one input of every pair, as shown in Fig. 6.28. These extra gates convert the duplication code into a 1-out-of-2 code. They will not affect the self checking ability of each design and will be automatically tested to meet the conditions of theorem 6.3 from codewords (every inverter input receives a 1 and a 0 from codewords).

6.7 : A TSC PARITY CODE CHECKER

A TSC parity checker is a circuit which converts a parity code to a 1-out-of-2 code. This is achieved by splitting the bits of a parity encoded word into two groups, where each group contains at least one bit, as follows:

1) Odd Parity : An odd number of 1's in the codeword creates an odd number of 1's in one group and an even number of 1's in the other. Therefore, if each group is checked for an odd number of 1's using an EXOR tree, one output will be high and the other low; i.e. a 1-out-of-2 code. Fig. 6.29a shows the resulting odd parity checker.

2) Even Parity : An even number of 1's in the codeword creates an odd number of 1's in both groups or an even number of 1's in both groups. Therefore, if one group is checked for an odd number of 1's using an EXOR tree and the other group is checked for an even number of 1's, then a 1-out-of-2 code output will be produced. Fig 6.29b shows the resulting even parity checker.

An independent subcircuit for each of the two outputs ensures that the checker is fault secure, so, again, only its self testing ability needs to be evaluated.

Example 6.5 : Consider an 8-bit data word with a single odd parity bit. Fig. 6.30a shows the 9-bit checking circuit where bits x_1, x_2, x_3 and x_4 form group 1 and bits x_5, x_6, x_7, x_8 and x_9 form group 2, such that;

$$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$y_2 = x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9 \qquad (6.11)$$

The data word is assumed to take on all possible combinations (2^8) so that the group 1 inputs will do likewise. The overall codeword, however, only takes on 2^8 out of the 2^9 possible combinations. (The other 2^8 combinations are even parity codewords.) The group 2 inputs, which include the parity bit, will take on all combinations, though, since the group 1 inputs are always able to provide appropriate levels to form an odd parity codeword. The truth tables are thus given in Fig. 6.30b and 6.30c for each subcircuit with all input combinations. They indicate that each gate will receive all its input combinations, so the conditions of corollary 6.1 are satisfied. Thus each subcircuit is self checking and the checker is TSC overall.

Example 6.5 demonstrates that a TSC checker for a k-bit parity encoded word can be constructed from a p-input EXOR tree and a q-input EXOR tree, such that:

$p + q = k$, where p and $q \geq 1$ (6.12)

In example 6.5 k=9, p=4 and q=5. The parity checker used as part of the partially self checking circuit in Fig. 5.21 is another example, where p=1 and q = k-1.

6.8 : A TSC 1-OUT-OF-N CODE CHECKER

A particular case of m-out-of-n codes are 1-out-of-n codes. They occur frequently in control signals where the code is generated from an m to n line decoder of the form of Fig. 6.31. An example of this is address decoding in a microprocessor based system. A TSC means of checking a 1-out-of-n code is therefore required. The author has investigated many 2-level AND/OR structures for this purpose, but all fail to be self testing because of only n codewords in 2^n combinations and the fact that each codeword has only a single 1. A more complex circuit must therefore be necessary.

Anderson [6.6] and Marouf [6.7] have proposed that a 1-out-of-n code is converted to a k-out-of-2k code, which is then checked by a TSC k-out-of-2k to 1-out-of-2 converter, as shown in Fig. 6.32a. Kraft [6.8] and Khakbaz [6.9] have proposed that a 1-out-of-n code is converted to a k x 1-out-of-2 code, which is then checked using the circuit described in section 6.6 and shown in Fig. 6.32b. It is this latter approach which is adopted here, as follows.

The 1-out-of-n to k x 1-out-of-2 converter, circuit L in Fig. 6.32b, has n inputs, $x_1 \ldots x_i \ldots x_n$ and k output pairs $\langle c_1 d_1 \rangle \ldots \langle c_j d_j \rangle \ldots \langle c_k d_k \rangle$, where k is given by:

$$k = \lceil \log_2 n \rceil \qquad (6.13)$$

A input codeword with x_i=1 maps to an output combination such that $\langle c_1 \ldots c_j \ldots c_k \rangle$ is the binary representation of

i. In each case d_j is the inverse of c_j. The only exception to this is where n is a power of 2, in which case the codeword with x_n=1 maps to c_1=...=c_j=...=c_k=0. Each output c_j and d_j is computed using an independent subcircuit, so the network is inherently fault secure. The network must also be code disjoint and self testing. These properties are demonstrated in example 6.6 below.

The k x 1-out-of-2 to 1 x 1-out-of-2 converter, circuit C in Fig. 6.32b, consists of a series of cascaded 2-input Morphic AND gates with level merging, if appropriate, as described in section 6.6. If n is not a power of 2, then all possible codewords will not be generated at the outputs of circuit L, since L has n code inputs and n code outputs. Circuit C implemented as a 2-level AND/OR (or OR/AND structure) will require all 2^k codewords to be self testing, so this design is not viable. However, every 2-input Morphic AND gate also requires all four codewords to be self testing, so the connections from circuit L to circuit C will have to ensure that this occurs. Example 6.6 illustrates this point.

Example 6.6 : Consider the 2 to 4 line decoder shown in Fig. 6.33. This is a similar circuit to that used in example 5.6, except that it has an additional input, enable g. When g=0 the decoder operates normally, but when g=1 all outputs are at 0. All outputs at 0 is not a 1-out-of-4 codeword, but if g is included as part of the output, a 1-out-of-5 code will always be generated.

Circuit L (of Fig 6.32b) will have five inputs, so that n=5, and k output pairs, where k=3 from (6.13). Fig. 6.34a gives the code inputs and required code outputs for this circuit, based on the procedure given above. Fig. 6.34b shows the Karnaugh maps for each output pair $\langle c_j d_j \rangle$, with noncode inputs assigned to give a ⟨00⟩ or ⟨11⟩ output. From these:

$c_1 = x_5 + x_4$, : $d_1 = x_3 + x_2 + x_1$

$$c_2 = x_3 + x_2 \quad : \quad d_2 = x_5 + x_4 + x_1$$
$$c_3 = x_5 + x_3 + x_1 \quad : \quad d_3 = x_4 + x_2 \qquad (6.14)$$

Applying corollary 6.2 to each Karnaugh map demonstrates that all subcircuits are self testing, so that L is TSC.

Circuit C (of Fig 6.32b) will be a 3-input Morphic AND gate consisting of two cascaded 2-input Morphic AND gates, as shown in Fig. 6.35a. Fig. 6.34a shows that only five of the possible eight output codewords are generated from circuit L, so that a Morphic AND gate with inputs $\langle c_1 d_1 \rangle$ and $\langle c_2 d_2 \rangle$ will not be self testing, since it will not receive $\langle 1010 \rangle^*$ as a codeword. The pairing of $\langle c_2 d_2 \rangle$ and $\langle c_3 d_3 \rangle$, however, does generate all four codewords, so these must therefore be connected to the first level Morphic AND gate in C and $\langle c_1 d_1 \rangle$ connected to the second level Morphic AND gate in C, as shown in Fig. 6.35a. Hence:

$$a_1 = c_2 \quad : \quad b_1 = d_2$$
$$a_2 = c_3 \quad : \quad b_2 = d_3$$
$$a_3 = c_1 \quad : \quad b_2 = d_1 \qquad (6.15)$$

Fig. 6.35b demonstrates that the second level Morphic AND gate in C receives all codewords, so C is self testing and the overall checker TSC. If the 3-input Morphic AND gate required for C was implemented as a 2-level AND/OR structure (equation 6.8), then it would not be self testing, as section 6.6 has already shown that it requires all input codewords for this purpose.

Anderson concluded from his design procedure [6.6] that a TSC code checker could not be constructed for 1-out-of-3 or 1-out-of-7 codes. He observed that a 1-out-of-7 code had too many codewords to be converted to a 2-out-of-4 code and if converted to a 3-out-of-6 code did not provide enough codewords to make the 3-out-of-6 checker self testing. Reddy [6.10] made the latter option possible by designing a 3-out-of-6 checker which required less code-

* $\langle c_1 d_1 c_2 d_2 \rangle$

words to be self testing. A 1-out-of-3 code checker cannot be designed using the method presented here either, since it has insufficient codewords to test even a 2-input gate. David [6.11] has overcome this problem by converting a 1-out-of-3 code to a 1-out-of-4 code using a TSC sequenial circuit and checking that.

Another 1-out-of-n code checker is now proposed, similar to that previously described in example 5.6. The network in Fig. 5.26, which is self testing only, can be TSC by replacing the two OR gates in the checker with EXOR gates.

Example 6.7 : Consider again the 2 to 4 line decoder with enable as in Fig. 6.33. Fig. 5.27 has detailed the effect of every single stuck-at fault within the circuit of Fig. 5.22. Fig. 6.36 uses this information to summarise the effects of every single stuck-at fault within the circuit of Fig. 6.33. It indicates that these faults can cause either an all 0 output or an output with two 1's, but never an incorrect code output.

Two EXOR gates are added to the circuit of Fig. 6.33 to form an output $\langle e_1 e_2 \rangle$, such that:

$$e_2 = g \oplus y_4 \oplus y_3$$
$$\text{and} \quad e_1 = y_2 \oplus y_1. \qquad (6.16)$$

Fig 6.37 details the complete truth table for these additional gates. It demonstrates that the EXOR checker is self testing for code inputs, but is not code disjoint. All input combinations to it which have an odd number of 1's produce a code output. This is not surprising since the circuit is the parity checker of section 6.7. However, on the basis of single stuck-at faults, a decoder output combination with three or five 1's cannot occur. The checker has an independent gate for each output, so it is effectively TSC. Note that the same is not true for unidirectional faults.

6.9: A TSC PERIODIC SIGNAL CHECKER

So far, only checkers for signals encoded in an error detecting code have been examined. There are also other signals in a microprocessor based system which need to be checked, notably single or multiphase clocks. The self checking synchronous machine design of section 5.8 also requires a checked clock signal.

Moreira de Souza et al in their design of a research oriented microcomputer [6.12] used a fault tolerant clock [6.13], derived from the work of Daly [6.14]. For the Electronic Switching System processor, Chang et al [6.15] detected variations in clock period by converting the signals to DC levels through low pass RC circuits and comparing these levels with known references. They checked for phase overlap with logic which could be tested via maintenance control inputs, as shown in Fig. 6.38. Usas [6.16] has proposed a TSC checker for single phase periodic signals and this is described more rigorously here.

A single phase clock signal is not encoded in space but encoded in time via three parameters; t_p (period), t_{on} and t_{off} (mark-space ratio), where $t_p = t_{on} + t_{off}$. A clock signal must be checked for these parameters, illustrated in Fig. 6.39, as well as stuck-at faults. This is accomplished with the circuit of Fig. 6.40. It consists of two monostables M1 and M2. M1 is triggered by the rising edge of the clock and should be set to generate a high level pulse of expected width t_{on}, whereas M2 is triggered by the falling edge of the clock and set to generate a pulse of width t_{off}. The two monostable outputs form a constantly changing 1-out-of-2 code during normal operation, as shown in Fig. 6.41b.

The parameters t_p, t_{on} and t_{off} can each remain constant, increase or decrease, so there are $3^3-1 = 26$ faulty combinations or non codewords. Some of these are

impossible faults such as t_p constant with increased t_{on} and t_{off}. Fig. 6.41 details the possible fault combinations and their effects on the monostable outputs. The clock signal SA1 or SA0 is equivalent to an infinite increase in t_{off} or t_{on} respectively, i.e. special cases of faults 3 and 5 in Fig 6.41e and 6.41g respectively. Fig 6.41 demonstrates that the checker produces a non code output for all possible fault combinations, so it is code disjoint. However, a permanent fault indication only occurs for stuck-at faults.

Each output uses an independent monostable, so the checker is fault secure for all faults affecting a single monostable, as well as faults resulting in a unidirectional output error. The self testing property cannot be determined without reference to the circuit used for the monostable. For example, a fault in M1 that causes its input to be fed directly to its output will not be detected and it would then prevent the clock SA1 failure from being detected. Usas describes appropriate monostables to ensure that the checker is self testing and hence TSC. He also details several applications for the checker.

More complex periodic waveforms, including multiphase clocks, can be generated by a self checking sequential machine, or by logic with self checking checkers monitoring the resultant waveform(s) for perhaps stuck-at failures or phase overlap. The input to both of these circuits should be a single phase clock checked in the manner described above.

6.10 : LITERATURE REVIEW

Carter et al [6.2] proposed the 2-input Morphic AND gate and its extension for n-inputs. Anderson [6.3] developed this theory and showed how the number of gate levels could be reduced when cascading 2-input Morphic AND gates. In addition, he discussed m-out-of-n checkers, presenting a

TSC 3-out-of-6 checker, which he used again for 1-out-of-n checkers [6.6].

There has been and continues to be a considerable amount written on the design of m-out-of-n checkers, either specifically [6.7,6.10,6.17-6.21], or as part of a wider discussion of TSC circuits [6.22,6.23]. These papers have generally resulted from researchers striving to reduce the hardware required for such checkers (the number of gates, gate inputs and gate levels), in order to achieve an increased speed of operation, reduced costs and fewer tests. Although 1-out-of-n codes are a subset of m-out-of-n codes, checkers specifically for them have also been considered [6.8,6.9]. A TSC checker for a 1-out-of-3 code has not yet been designed using combinational logic, although David [6.11] has proposed a TSC sequential machine for this purpose.

There has been little added to the work of Carter and Anderson for TSC n-bit comparators using Morphic AND gates (also known as 1-out-of-2 checkers), other than their implementation in specific technologies. These comparaors usually compare two n-bit vectors, but Hughes [6.24, 6.25] has extended the principles involved to comparators for more than two input vectors.

There has also been no alternative proposals to the EXOR trees for TSC parity checkers [6.2,6.26,6.47]. What is discussed though is their application. A number of authors have studied TSC checkers for combinational circuits which have uncoded inputs or outputs [6.27-6.30] and this often involves the use of parity predicion.

In addition to the codes mentioned so far, TSC checkers have also been proposed for Berger codes [6.31,6.32], separable codes in general [6.33,6.34] and low cost arithmetic codes [6.35]. The construction of all these checkers using programmable logic arrays has been widely investigated [6.36-6.38], as well as their implementation

in various technologies [6.39-6.43].

Other designs include the TSC periodic signal checker described above [6.16], a self checking linear feedback register [6.44], a self testing only decoder circuit [6.45] which was redesigned to be TSC in a later paper [6.46], a TSC linear counter [6.47] and a self testing arbiter circuit for multi-microcomputer systems [6.48].

6.11 : REFERENCES

6.1) Logical design of switching systems - D. Lewin; Thomas Nelson and Sons Ltd, England; Second edition, 1974; Chapter 3.

6.2) Design of dynamically checked computers. - W. C. Carter, P. R. Schneider; Proc. IFIPS Congress, Vol. 2; Edinburgh; August 1968; North-Holland Publishing Company, Amsterdam (1969); pp. 878-883.

6.3) Design of self-checking digital networks using coding techniques - D. A. Anderson; Coordinated Science Laboratory Report R-527; September, 1971; University of Illinois, Urbana, Illinois, USA.

6.4) Private communication with W. C. Carter; May, 1983.

6.5) A theory of design of fault-tolerant computers using standby sparing - W. C. Carter, W. G. Bouricius, D. C. Jessep, J. P. Roth, P. R. Schneider, A. B. Wadia; FTCS-1*; pp. 83-86.

6.6) Design of totally self-checking check circuits for m-out-of-n codes; D. A. Anderson, G. Metz; IEEE Trans. Comput.; Vol. C-22, No. 3; March, 1973; pp. 263-269.

6.7) Efficient design of self-checking checkers for m-out-of-n codes - M. A. Marouf, A. D. Friedman; FTCS-7*: pp. 143-149.

6.8) Microprogrammed control and reliable design of small computers - G. D. Kraft, W. N. Toy; Prentice-Hall Inc., Englewood Cliffs,Jersey, USA; 1981; pp. 238-264.

6.9) Totally self-checking checker for 1-out-of-n code using two-codes - J. Khakbaz; IEEE Trans. Comput.; Vol. C-31, No. 7; July, 1982; pp. 677-681.

6.10) A note on self-checking checkers - S. M. Reddy; IEEE Trans. Comput.; October, 1974; pp. 1100-1102.

6.11) A totally self-checking 1-out-of-3 checker - Rene David; IEEE Trans. Comput., Vol. C-27, No. 6; June, 1978; pp. 570-572.

6.12) A research oriented microcomputer with built in auto-diagnostics - J. Moreira de Souza, E. Peixoto Paz, C. Landrault; FTCS-6*; pp. 3-8.

6.13) Fault-tolerant digital clocking system - J. Moreira de Souza, E. Peixoto Paz; Electronics Letters; Vol. 11, No. 18; September 4, 1975; pp. 433-434.

6.14) A fault tolerant digital clocking system - W. M. Daly, A. L. Hopkins, J. F. McKenna; FTCS-3*; pp.

17-21.

6.15) Maintenance techniques of a microprogrammed self-checking control complex of an electronic switching system - H. Y. P. Chang, G. W. Heimbigner, D. J. Senese, T. L. Smith; IEEE Trans. Comput., Vol. C-22, No. 5; May, 1973; pp. 501-512.

6.16) A totally self-checking checker design for the detection of errors in periodic signals - A. M. Usas; IEEE Trans. Comput.; Vol. C-24, No.5; May, 1975; pp. 483-489.

6.17) A new design method for m-out-of-n TSC checkers - N. Gaitanis, C. Halatsis; IEEE Trans. Comput.; Vol. C-32, No. 3; March 1983; pp. 273-283.

6.18) Fast and efficient totally self-checking checkers for m-out-of (2m+/-1) codes - C. Halatsis, N. Gaitanis, M. Sigala; IEEE Trans. Comput.; Vol. C-32, No. 5; May, 1983; pp. 507-511.

6.19) Modular realization of totally self checking checkers for m-out-of-n codes - C. Efstathiou, C. Halatsis; FTCS-13*; pp. 154-161.

6.20) Design method of totally self checking checkers for m-out-of-n codes; S. Piestrak;FTCS-13*;pp. 162-168.

6.21) A 3-level realization of totally self-checking checkers for m-out-of-n codes - T. Nanya, Y. Tohma; FTCS-13*; pp. 173-176.

6.22) A theory of design of fault-tolerant computers using standby sparing - W. C. Carter, W. G. Bouricius, D. C. Jessep, J. P. Roth, P. R. Schneider, A. B. Wadia; IBM Research Yorktown; RC 3261; February, 1971; pp. 1-11.

6.23) Diagnosis and reliable design of digital systems - M. A. Breuer, A. D. Friedman; Pitman Publishing Ltd., London, 1977; pp. 265-272.

6.24) Design of totally self-checking comparators with an arbitrary number of inputs - J. L. A. Hughes, E. J. McCluskey, D. J. Lu; IEEE Trans. Comput.; Vol. C-33, No. 6; June, 1984; pp. 546-550.

6.25) Design of totally self-checking comparators with an arbitrary number of inputs - J. L. A. Hughes, E. J. McCluskey, D. J. Lu; FTCS-13*; pp.169-172.

6.26) Self-testing embedded parity trees - J. Khazbaz; FTCS-12*; pp. 109-116.

6.27) A self-testing group-parity prediction checker and its use for built-in testing - E. Fujiwara, N. Mutoh, K. Matsuoka; IEEE Trans. Comput.; Vol. C-33,

No.6, June, 1984; pp. 578-583.

6.28) A totally self-checking generalized prediction checker and its use for built-in testing - E. Fujiwara, K. Matsuoka; FTCS-15*; pp. 384-389.

6.29) The design of totally self-checking embedded checkers - N. K. Jha, J. A. Abraham; FTCS-14*; pp. 265-270.

6.30) A self-testing group-parity prediction checker and its use for built-in testing - E. Fujiwara; FTCS-13*; pp. 146-153.

6.31) Design of fast self-testing checkers for a class of Berger codes - S. J. Piestrak;FTCS-15*;pp. 418-423.

6.32) Design of self-checking checkers for Berger codes - M. A. Marouf, A. D. Friedman;FTCS-8*;pp. 179-184.

6.33) On totally-self-checking checkers for separable codes - M. J. Ashjaee, S. M. Reddy; FTCS-7*; pp. 151-156.

6.34) On totally-self-checking checkers for separable codes - M. J. Ashjaee, S. M. Reddy; IEEE Trans. Comput.; Vol C-26, pp. 737-744.

6.35) Totally self checking checkers for low cost arithmetic codes - N. Gaitanis; FTCS-14*; pp. 260-264.

6.36) Design and application of self-testing comparators implemented with MOS PLA's - Y. Tamir, C. H. Sequin; IEEE Trans. Comput.; Vol. C-33, No. 6; June, 1984; pp. 493-506.

6.37) PLA implementation of k-out-of-n code TSC checker - B. Bose, D. J. Lin; IEEE Trans. Comput.; Vol. C-33, No. 6; June, 1984; pp. 583-588.

6.38) The design of totally self-checking circuits using programmable logic arrays - S. L. Wang, A. Avizienis; FTCS-9*; pp. 173-180.

6.39) Multivalued I^2L circuits for TSC checkers - D. Etiemble; FTCS-9*; pp. 181-184.

6.40) Threshold I^2L totally self checking circuits - Y. W. Yang, S. M. Liu, T. C. Chen; FTCS-10*; pp. 284-287.

6.41) Analysis of a class of totally self-checking functions implemented in a MOS LSI general logic structure - M. W. Sievers, A. Avizienis; FTCS-11*; pp. 256-261.

6.42) Totally self-checking CMOS circuits using a hybrid relalization - N. K. Jha, J. A. Abraham; FTCS-15*;

pp. 154-158.

6.43) Techniques for efficient MOS implementation of totally self-checking checkers - N. K. Jha, J. A. Abraham; FTCS-15*; pp. 430-435.

6.44) Self-checking linear feedback shift registers - D. J. Lu; FTCS-10*; pp. 269-271.

6.45) A simple self-testing decoder checking circuit - W. C. Carter, K. A. Duke, D.C. Jessep; IEEE Trans. Comput.; November, 1971; pp. 1413-1415.

6.46) Theory and use of checking circuits - W. C. Carter; Infotech*; pp. 415-454.

6.47) Error-correcting codes and self-checking circuits - D. K. Pradhan, J. J. Stiffler; Computer; March, 1980; pp. 27-37.

6.48) A self-testing arbiter circuit for multimicrocomputer systems - M. Courvoisier, J. C. Geffroy, J. P. Seck; FTCS-10*; pp. 281-283.

* see section B.5.

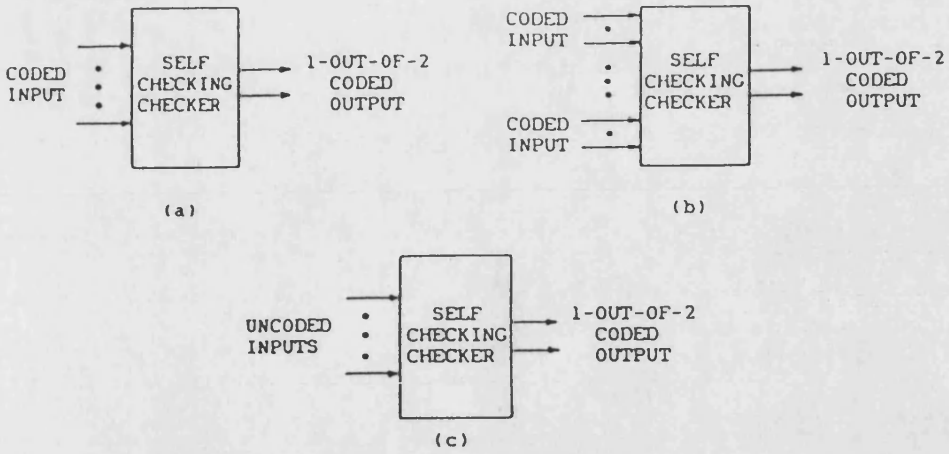


FIGURE 6.1 SELF CHECKING CIRCUITS

INPUTS		STUCK-AT FAULTS DETECTED		
a	b	a	b	c
0	0	-	-	-
0	1	1	-	1
1	0	-	1	1
1	1	0	0	0

- = no fault detected
1 = SA1 fault detected
0 = SA0 fault detected

FIGURE 6.2 FAULT TABLE FOR 2-INPUT AND GATE

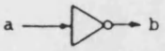
INPUTS		STUCK-AT FAULTS DETECTED		
a	b	a	b	c
0	0	1	1	1
0	1	-	0	0
1	0	0	-	0
1	1	-	-	-

FIGURE 6.3 FAULT TABLE FOR 2-INPUT OR GATE

INPUTS			STUCK-AT FAULTS DETECTED			
a	b	c	a	b	c	d
0	0	0	-	-	-	1
0	0	1	-	-	-	1
0	1	0	-	-	-	1
0	1	1	1	-	-	1
1	0	0	-	-	-	1
1	0	1	-	1	-	1
1	1	0	-	-	1	1
1	1	1	0	0	0	0

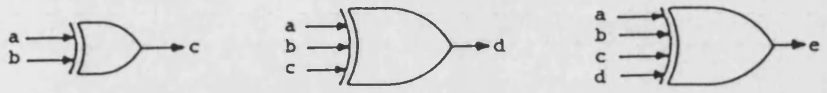
INPUTS			STUCK-AT FAULTS DETECTED			
a	b	c	a	b	c	d
0	0	0	1	1	1	1
0	0	1	-	-	0	0
0	1	0	-	0	-	0
0	1	1	-	-	-	0
1	0	0	0	-	-	0
1	0	1	-	-	-	0
1	1	0	-	-	-	0
1	1	1	-	-	-	0

FIGURE 6.4 FAULT TABLES FOR 3-INPUT GATES



INPUT	STUCK-AT FAULTS DETECTED	
a	a	b
0	1	0
1	0	1

FIGURE 6.5 FAULT TABLE FOR INVERTER

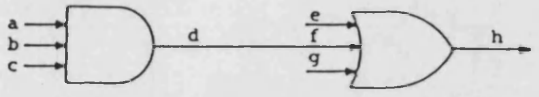


INPUTS		STUCK-AT FAULTS DETECTED		
a	b	a	b	c
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

INPUTS			STUCK-AT FAULTS DETECTED			
a	b	c	a	b	c	d
0	0	0	1	1	1	1
0	0	1	1	1	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	0	1	1
1	1	1	0	0	0	0

INPUTS				STUCK-AT FAULTS DETECTED				
a	b	c	d	a	b	c	d	e
0	0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	0	0
0	0	1	0	1	1	0	1	0
0	0	1	1	1	1	0	0	1
0	1	0	0	1	0	1	1	0
0	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1
0	1	1	1	1	0	0	0	0
1	0	0	0	0	1	1	1	0
1	0	0	1	0	1	1	0	1
1	0	1	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	0	0	0	1	1	1
1	1	0	1	0	0	1	0	0
1	1	1	0	0	0	0	1	0
1	1	1	1	0	0	0	0	1

FIGURE 6.6 FAULT TABLES FOR EXCLUSIVE-OR GATES

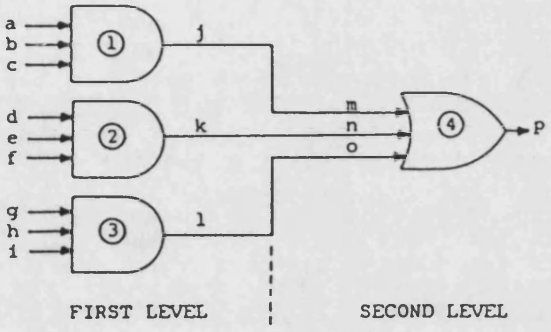


(a)

AND INPUTS V t ∈ T&			FAULTS DETECTED	d (FAULT FREE)	d^f (d WITH FAULT)	EQUIVALENT FAULT IN	OR INPUTS TO PROPAGATE FAULT		
a	b	c				f	e	f	g
0	1	1	a,d SA1	0	1	SA1	0	0	0
1	0	1	b,d SA1	0	1	SA1	0	0	0
1	1	0	c,d SA1	0	1	SA1	0	0	0
1	1	1	a,b,c,d SA0	1	0	SA0	0	1	0
					ADDITIONAL OR INPUTS FOR ∀ t ∈ T+		1	0	0
							0	0	1

(b)

FIGURE 6.7 TESTING CASCADED GATES



(a) CIRCUIT

	TESTS FOR FIRST LEVEL GATES									SECOND LEVEL INPUTS TO PROPAGATE FAULT		
		①			②			③				
	a	b	c	d	e	f	g	h	i	m	n	o
V t ∈ T&	0	1	1		X			X		0	0	0
	1	0	1		X			X		0	0	0
	1	1	0		X			X		0	0	0
	1	1	1		X			X		1	0	0
X = OUTPUT AT 0		X		0	1	1		X		0	0	0
		X		1	0	1		X		0	0	0
		X		1	1	0		X		0	0	0
		X		1	1	1		X		0	1	0
12 CONDITIONS TO SATISFY		X			X		0	1	1	0	0	0
		X			X		1	0	1	0	0	0
		X			X		1	1	0	0	0	0
		X			X		1	1	1	0	0	1

(b) REQUIRED TESTS (12)

V t ∈ T+

X = OUTPUT AT 0

6 TESTS IN FULLY MERGED TEST SETS

a	b	c	d	e	f	g	h	i	m	n	o
0	1	1	0	1	1	0	1	1	0	0	0
1	0	1	1	0	1	1	0	1	0	0	0
1	1	0	1	1	0	1	1	0	0	0	0
1	1	1		X			X		1	0	0
	X		1	1	1		X		0	1	0
	X			X		1	1	1	0	0	1

a	b	c	d	e	f	g	h	i	m	n	o
0	1	1	1	1	0	1	0	1	0	0	0
1	1	0	1	0	1	0	1	1	0	0	0
1	1	0	1	0	1	0	1	1	0	0	0
1	1	1		X			X		1	0	0
	X		1	1	1		X		0	1	0
	X			X		1	1	1	0	0	1

(C) POSSIBLE MERGED TEST SETS

FIGURE 6.8 TESTING A 2-LEVEL AND/OR STRUCTURE

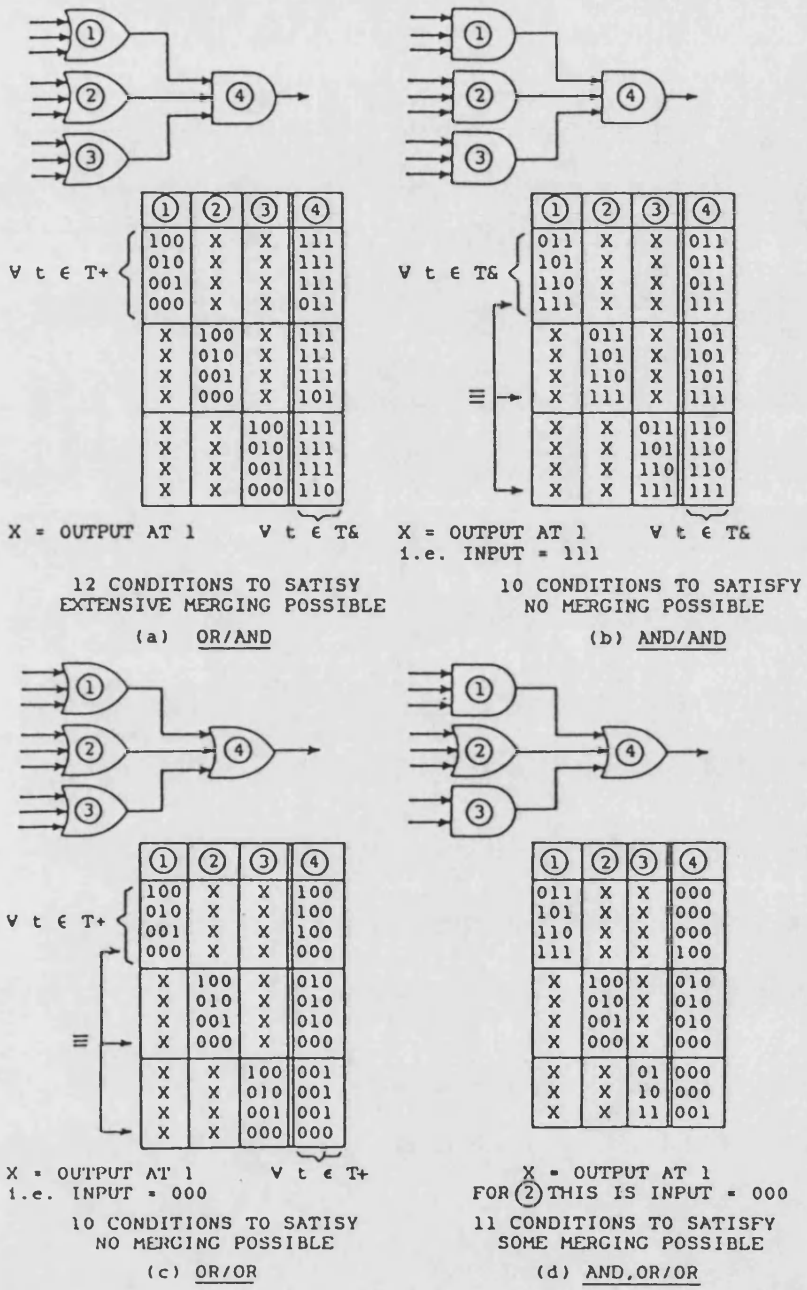


FIGURE 6.9 TESTING 2-LEVEL STRUCTURES

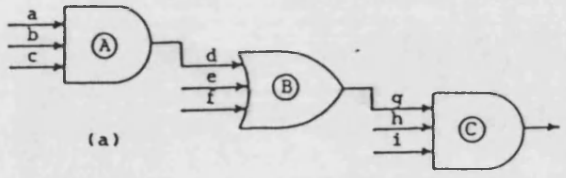
(a)

EXOR (1) INPUTS V t ∈ T ⊕ a	b	FAULTS DETECTED	c (FAULT FREE)	c^f (c WITH FAULT)	EQUIVALENT FAULT IN d	EXOR (2) INPUTS TO PROPAGATE FAULT d	e
0	1	a SA1,b SA0,c SA0	1	0	SA0	1	*
1	0	a SA0,b SA1,c SA0	1	0	SA0	1	*
1	1	a SA0,b SA0,c SA1	0	1	SA1	0	*

* = 0 OR 1, BUT REQUIRE V t ∈ T ⊕

(b)

FIGURE 6.10 TESTING CASCADED EXCLUSIVE-OR GATES



(a)

	A a	b	c	B d	e	f	C g	h	i	
1) START AT A V t ∈ T&	0	1	1	0	0	0	0	1	1	
	1	0	1	0	0	0	0	1	1	
	1	1	0	0	0	0	0	1	1	
	1	1	1	1	0	0	1	1	1	
2) START AT B V t ∈ T+		'1'		1	0	0	1	1	1	↵
		'0'		0	1	0	1	1	1	
		'0'		0	0	1	1	1	1	
		'0'		0	0	0	0	1	1	↵
3) V t ∈ T& FOR C		*			'0'		0	1	1	↵
		*			'1'		1	0	1	
		*			'1'		1	1	0	
		*			'1'		1	1	1	↵

↵ = TEST ALREADY EXISTS
'1' = OUTPUT AT 1, '0' = OUTPUT AT 0
* = OUTPUT AT 0 OR 1

(b)

FIGURE 6.11 TESTING MORE THAN TWO CASCADED GATES

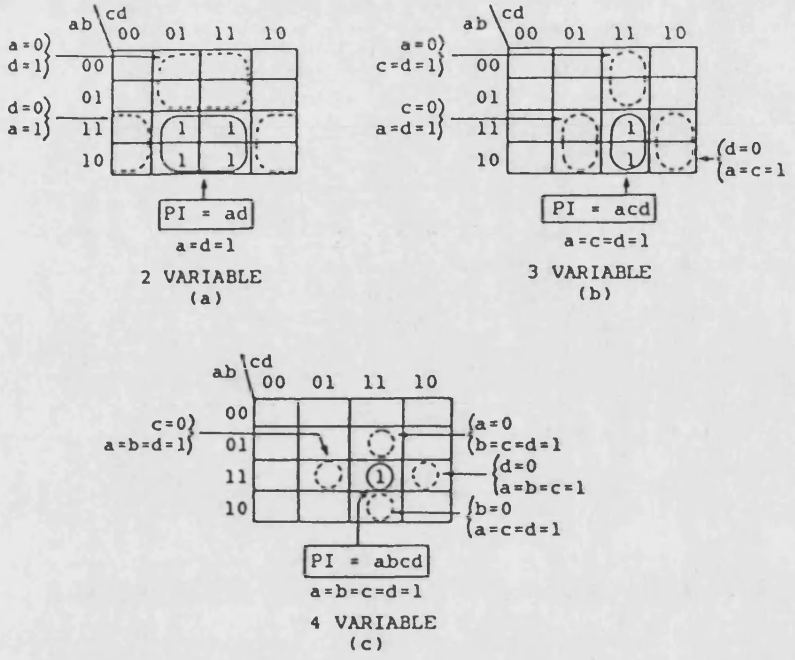


FIGURE 6.12 EXPANSION OF A PRIME IMPLICANT (PI)

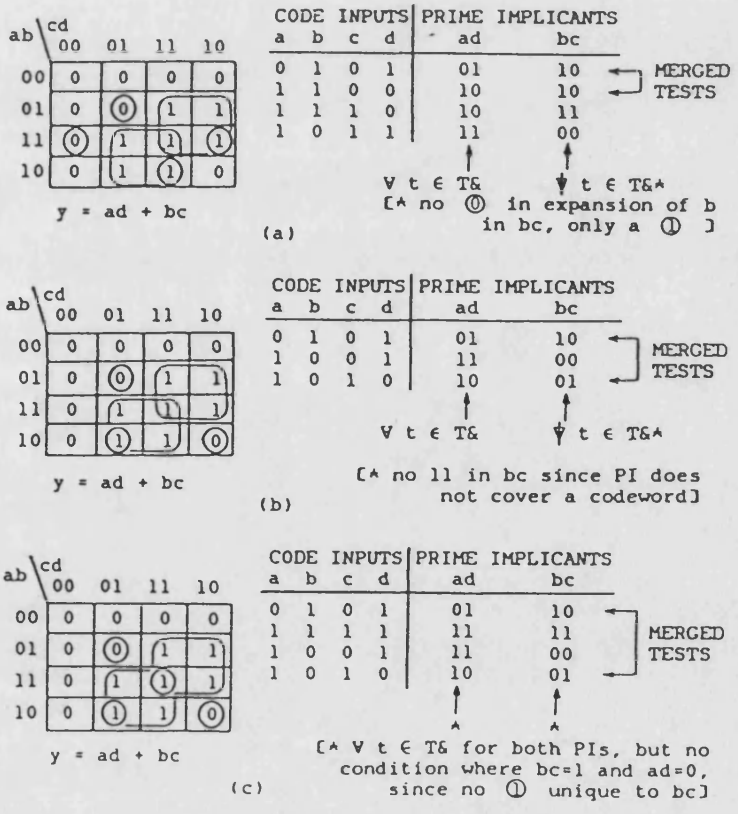


FIGURE 6.13 NON TSC CHECKER DESIGNS

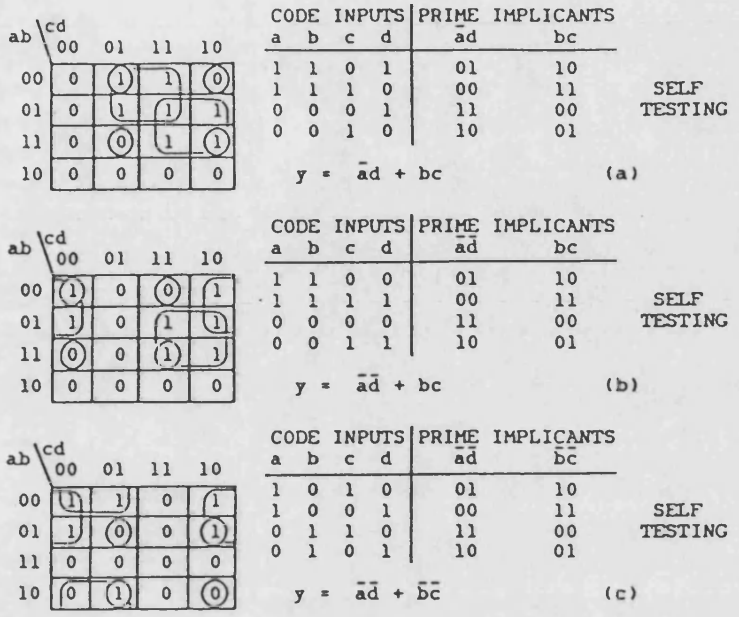


FIGURE 6.14 TSC CHECKER DESIGNS WITH INVERTED INPUTS

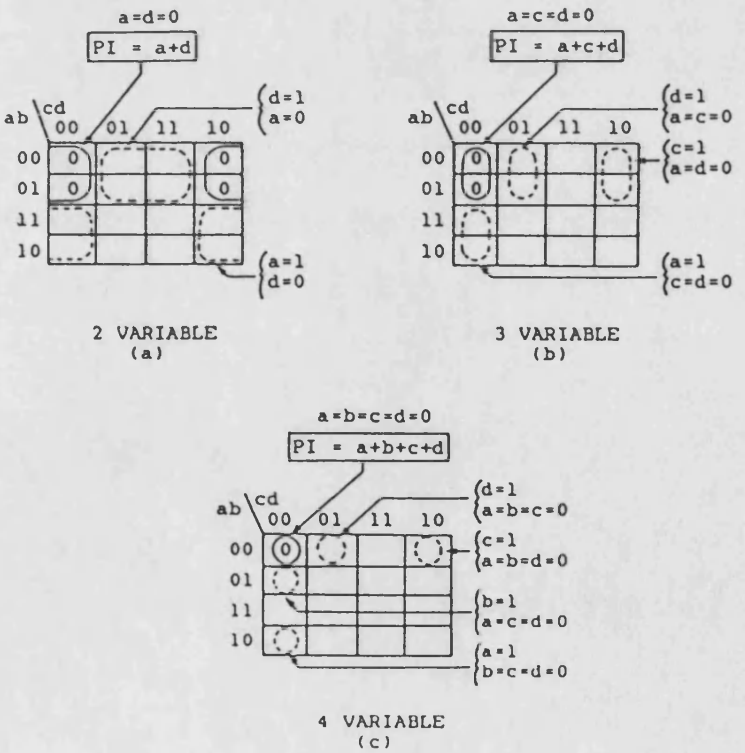


FIGURE 6.15 EXPANSION OF A PRIME IMPLICATE (PI)

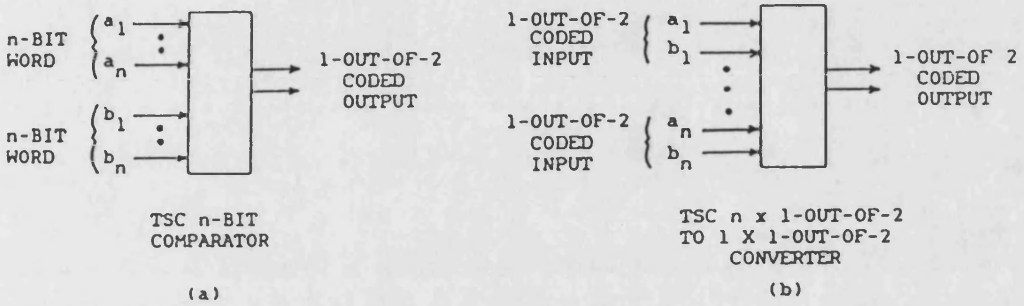
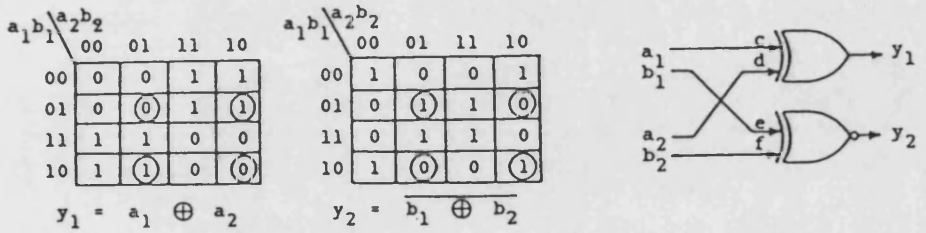


FIGURE 6.16 TSC COMPARATORS



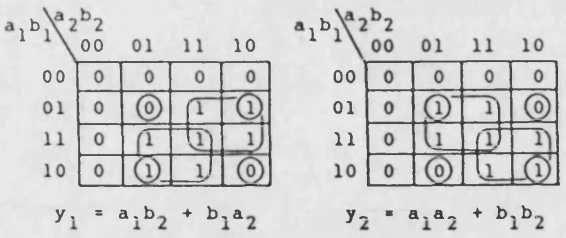
INPUTS				GATE	INPUTS	OUTPUTS	
a_1	b_1	a_2	b_2	cd	ef	y_1	y_2
0	0	0	0	00	00	0	1
0	0	0	1	00	01	0	0
0	0	1	0	01	00	1	1
0	0	1	1	01	01	1	0
0	1	0	0	00	10	0	0
0	1	0	1	00	11	0	1
0	1	1	0	01	10	1	0
0	1	1	1	01	11	1	1
1	0	0	0	10	00	1	1
1	0	0	1	10	01	1	0
1	0	1	0	11	00	0	1
1	0	1	1	11	01	0	0
1	1	0	0	10	10	1	0
1	1	0	1	10	11	1	1
1	1	1	0	11	10	0	0
1	1	1	1	11	11	0	1

CODE INPUTS

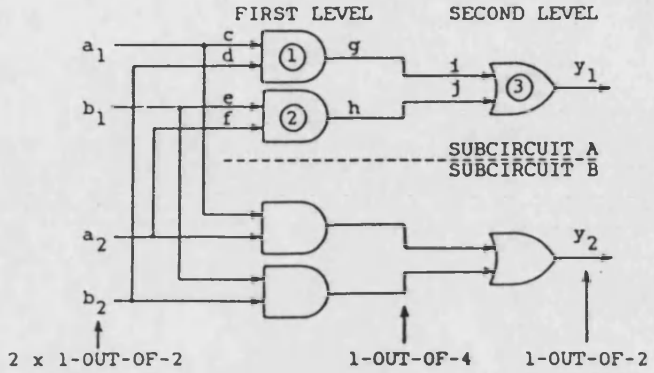
NON CODE INPUTS WITH CODE OUTPUTS

(c) TRUTH TABLE

FIGURE 6.17 2 x 1-OUT-OF-2 TO 1 x 1-OUT-OF-2 CONVERTER



(a) KARNAUGH MAPS



(b) CIRCUIT

①	②	③
01	X	00
10	X	00
11	X	10
X	01	00
X	10	00
X	11	01

X = OUTPUT AT 0
6 CONDITIONS TO SATISFY

(c) TESTING SUBCIRCUIT A

	INPUTS				GATE INPUTS			OUTPUT	
	a_1	b_1	a_2	b_2	① cd	② ef	③ ij	y_1	y_2
	0	0	0	0	00	00	00	0	0
	0	0	0	1	01	00	00	0	0
	0	0	1	0	00	01	00	0	0
	0	0	1	1	01	01	00	0	0
	0	1	0	0	00	10	00	0	0
CODE INPUTS →	0	1	0	1	01*	10*	00*	0	1
CODE INPUTS →	0	1	1	0	00*	11*	01*	1	0
	0	1	1	1	01	11	01	1	1
	1	0	0	0	10	00	00	0	0
CODE INPUTS →	1	0	0	1	11*	00*	10*	1	0
CODE INPUTS →	1	0	1	0	10*	01*	00*	0	1
	1	0	1	1	11	01	10	1	1
	1	1	0	0	10	10	00	0	0
	1	1	0	1	11	10	10	1	1
	1	1	1	0	10	11	01	1	1
	1	1	1	1	11	11	11	1	1

* INDICATES THE FULLY MERGED TEST SET FOR SUBCIRCUIT A

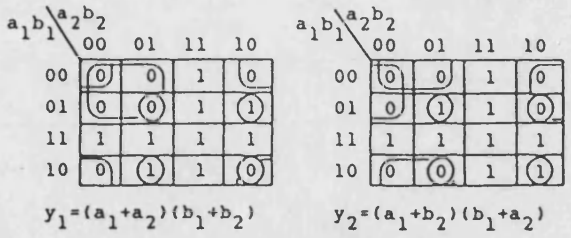
(d) TRUTH TABLE

CODE INPUTS				FAULT															
				c		d		e		f		g		h		i		j		y_1	
a_1	b_1	a_2	b_2	SA0	SA1	SA0	SA1	SA0	SA1	SA0	SA1	SA0	SA1	SA0	SA1	SA0	SA1	SA0	SA1	SA0	SA1
0	1	0	1		X				X		X		X		X		X		X		X
0	1	1	0					X		X			X					X		X	
1	0	0	1	X		X						X				X				X	
1	0	1	0				X		X				X		X		X		X		X

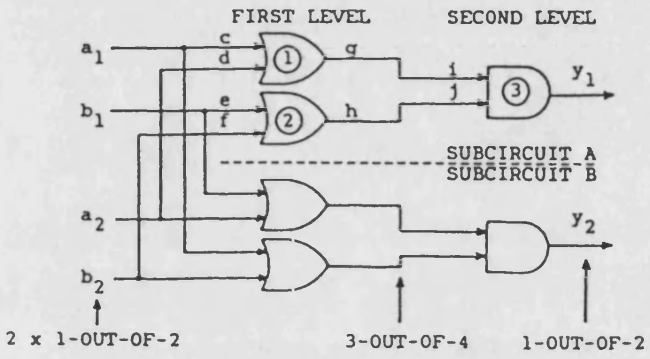
X INDICATES FAULT DETECTED

(e) FAULT TABLE FOR SUBCIRCUIT A

FIGURE 6.18 TSC 2-INPUT AND/OR MORPHIC AND GATE



(a) KARNAUGH MAPS



(b) CIRCUIT

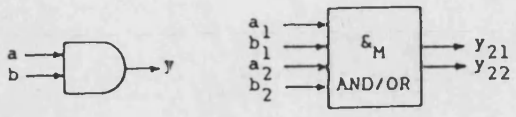
	INPUTS a_1	b_1	a_2	b_2	OUTPUT y_1	y_2
	0	0	0	0	0	0
	0	0	0	1	0	0
	0	0	1	0	0	0
	0	0	1	1	1	1
	0	1	0	0	0	0
CODE INPUTS	0	1	0	1	0	1
CODE INPUTS	0	1	1	0	1	0
	0	1	1	1	1	1
	1	0	0	0	0	0
CODE INPUTS	1	0	0	1	1	0
CODE INPUTS	1	0	1	0	0	1
	1	0	1	1	1	1
	1	1	0	0	1	1
	1	1	0	1	1	1
	1	1	1	0	1	1
	1	1	1	1	1	1

(c) TRUTH TABLE

INPUTS a_1	b_1	a_2	b_2	GATE INPUTS ① cd	② ef	③ ij	OUTPUT y_1
0	1	0	1	00	11	01	0
0	1	1	0	01	10	11	1
1	0	0	1	10	01	11	1
1	0	1	0	11	00	10	0

(d) TESTING SUBCRCUIT A

FIGURE 6.19 TSC 2-INPUT OR/AND MORPHIC AND GATE



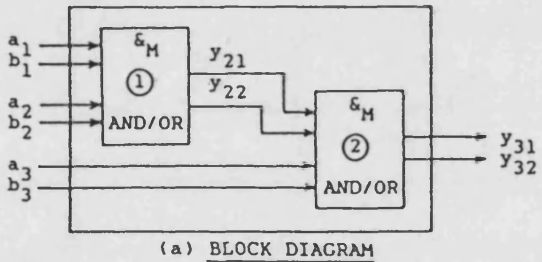
a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a_1	b_1	a_2	b_2	y_{21}	y_{22}
0	0	0	0	0	0
0	0	1	1	0	0
1	1	0	0	0	0
1	1	1	1	1	1
0	0	0	1	0	0
0	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	1	1
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	1	1
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1

'1' = 01 or 10
'0' = 00 or 11

CODE INPUTS

FIGURE 6.20 CONVENTIONAL AND MORPHIC AND GATES

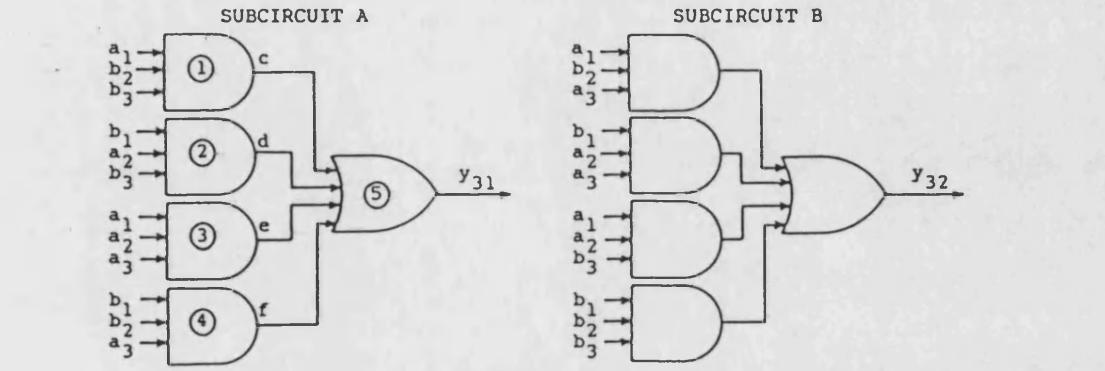


(a) BLOCK DIAGRAM

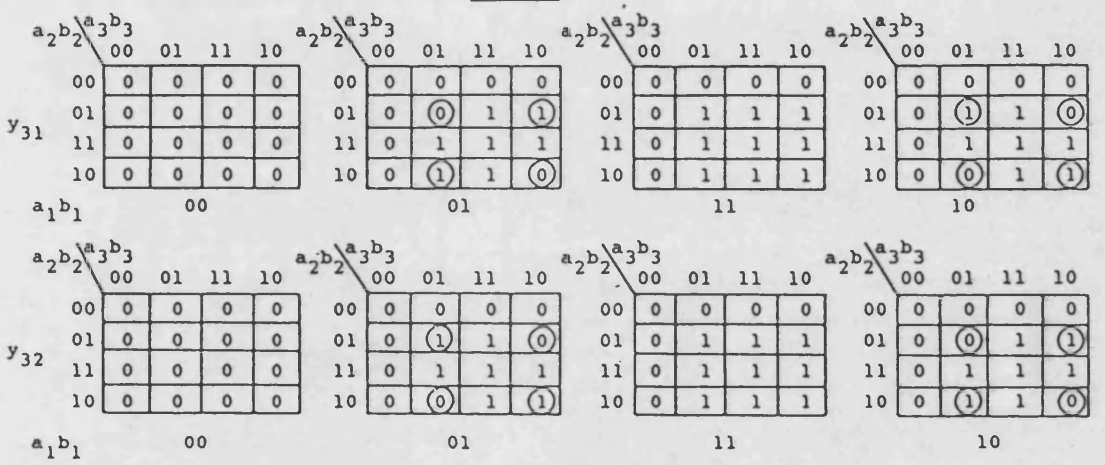
CODE INPUTS								OUTPUTS	
①				②					
a_1	b_1	a_2	b_2	y_{21}	y_{22}	a_3	b_3	y_{31}	y_{32}
0	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1	1	0
0	1	1	0	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1	0
1	0	0	1	1	0	1	0	0	1
1	0	1	0	0	1	0	1	0	1
1	0	1	0	0	1	1	0	1	0

(b) TRUTH TABLE

FIGURE 6.21 3-INPUT MORPHIC AND GATE



(a) CIRCUIT

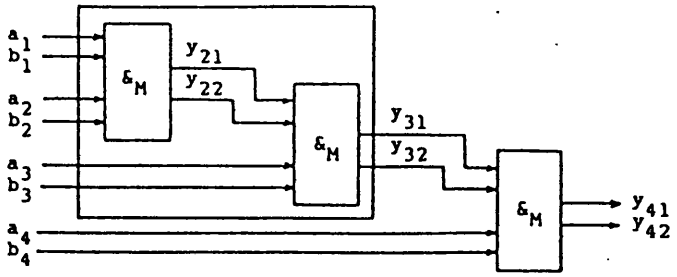


(b) KARNAUGH MAPS

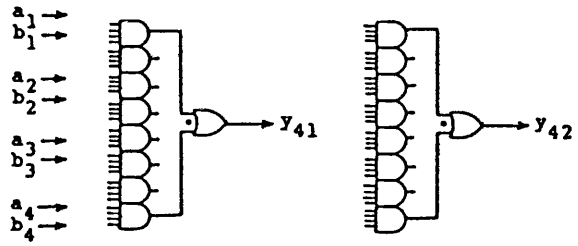
CODE INPUTS						GATE INPUTS												OUTPUT
						①			②			③			④			⑤				
a_1	b_1	a_2	b_2	a_3	b_3	a_1	b_2	b_3	b_1	a_2	b_3	a_1	a_2	a_3	b_1	b_2	a_3	c	d	e	f	y_{31}
0	1	0	1	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	1	0	0	0	0	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	0	1	1	1	1	0	1	0	1	0	0	0	1	0	0	1
0	1	1	0	1	0	0	0	0	1	1	0	0	1	1	1	0	1	0	0	0	0	0
1	0	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	0	1	0	0	0	1
1	0	0	1	1	0	1	1	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0
1	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	0	0	1	0	1	1	1	0	0	1	0	0	1	0	1

(c) TESTING SUBCIRCUIT A

FIGURE 6.22 3-INPUT AND/OR MORPHIC AND GATE

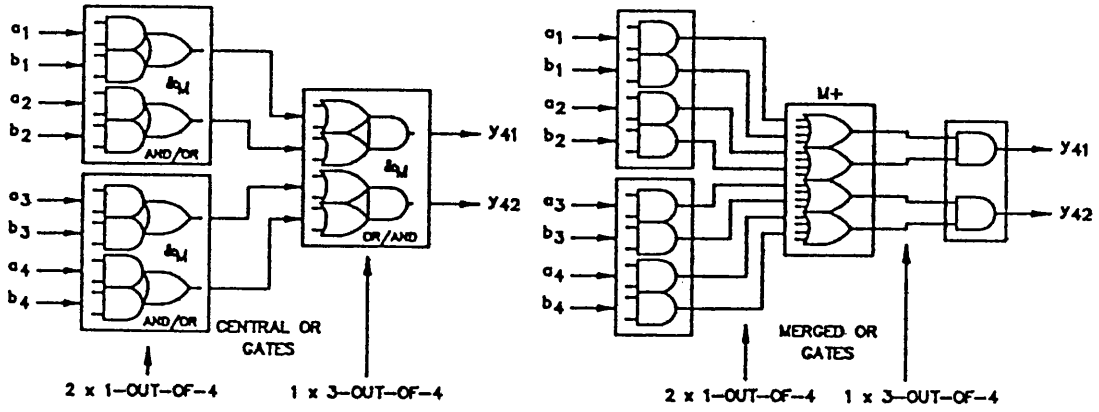


(a) CASCADED 2-INPUT MORPHIC AND GATES



(b) 2-LEVEL AND/OR STRUCTURE

FIGURE 6.23 4-INPUT MORPHIC AND GATES



(a) UNMERGED (b) MERGED

FIGURE 6.24 4-INPUT MORPHIC AND GATE WITH MERGED OR GATES

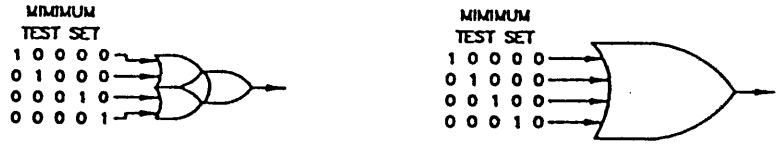


FIGURE 6.25 TESTING A 4-INPUT OR GATE

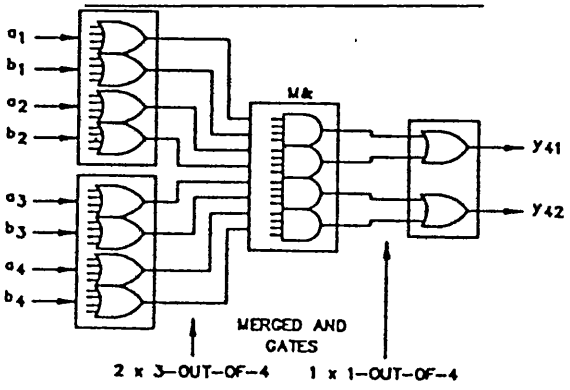


FIGURE 6.26 4-INPUT MORPHIC AND GATE WITH MERGED AND GATES

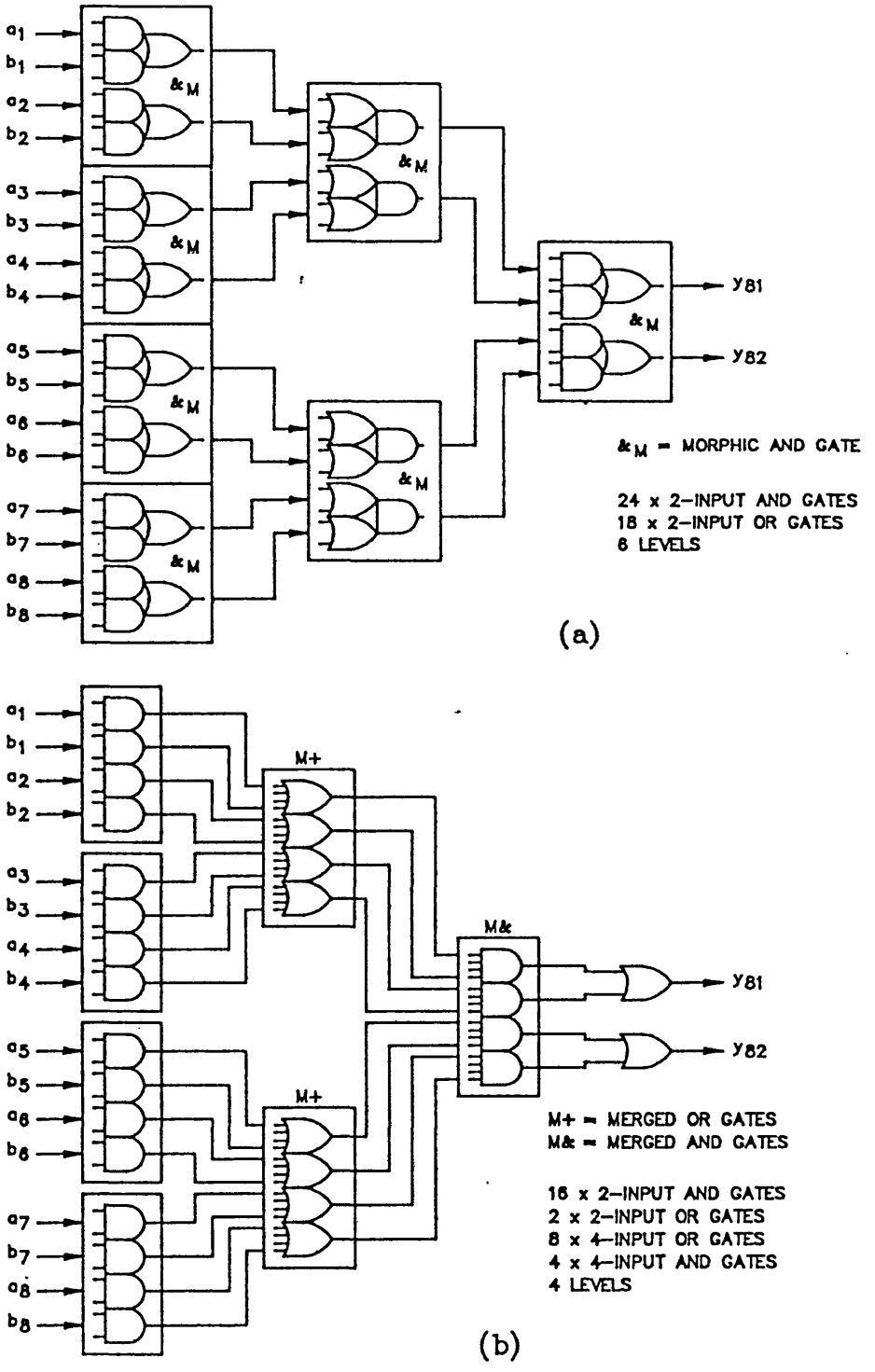


FIGURE 6.27 8-INPUT MORPHIC AND GATE

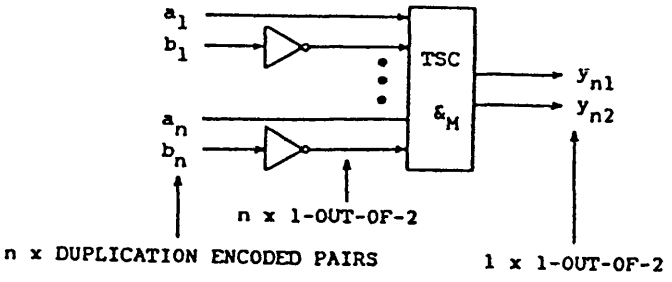
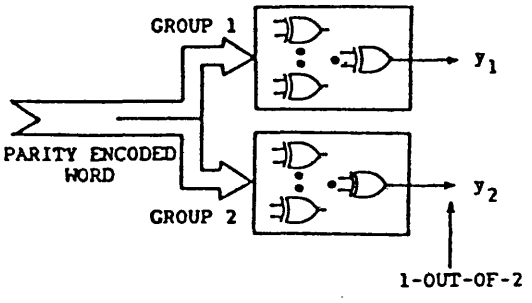
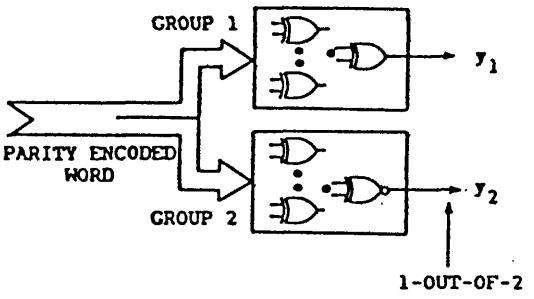


FIGURE 6.28 TSC COMPARATOR

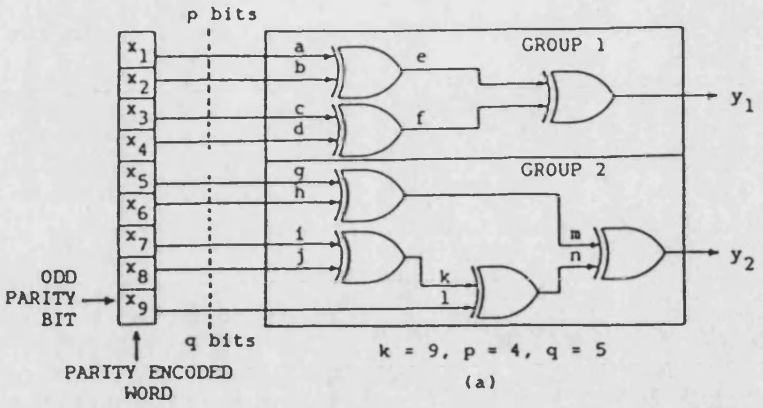


(a) ODD PARITY



(b) EVEN PARITY

FIGURE 6.29 TSC PARITY CHECKERS



GATE INPUTS						OUTPUT
a	b	c	d	e	f	y_1
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	0	0

∀ t ∈ T ⊕

(b) GROUP 1 TRUTH TABLE

GATE INPUTS												OUTPUT
g	h	i	j	k	l	m	n					y_2
0	0	0	0	0	0	0	0					0
0	0	0	0	0	1	0	1					1
0	0	0	1	1	0	0	1					1
0	0	0	1	1	1	0	0					0
0	0	1	0	1	0	0	1					1
0	0	1	0	1	1	0	0					0
0	0	1	1	0	0	0	0					0
0	0	1	1	0	1	0	1					1
0	1	0	0	0	0	1	0					1
0	1	0	0	0	1	1	1					0
0	1	0	1	1	0	1	1					0
0	1	0	1	1	1	1	0					1
0	1	1	0	1	0	1	1					0
0	1	1	0	1	1	1	0					1
0	1	1	1	0	0	1	0					1
0	1	1	1	0	1	1	1					0
1	0	0	0	0	0	1	0					1
1	0	0	0	0	1	1	1					0
1	0	0	1	1	0	1	1					0
1	0	0	1	1	1	1	0					1
1	0	1	0	1	0	1	1					0
1	0	1	0	1	1	1	0					1
1	0	1	1	0	0	1	0					1
1	0	1	1	0	1	1	1					0
1	1	0	0	0	0	0	0					0
1	1	0	0	0	1	0	1					1
1	1	0	1	1	0	0	1					1
1	1	0	1	1	1	0	0					0
1	1	1	0	1	0	0	1					1
1	1	1	0	1	1	0	0					0
1	1	1	1	0	0	0	0					0
1	1	1	1	0	1	0	1					1

∀ t ∈ T ⊕

(c) GROUP 2 TRUTH TABLE

FIGURE 6.30 TSC 9-BIT PARITY CHECKER

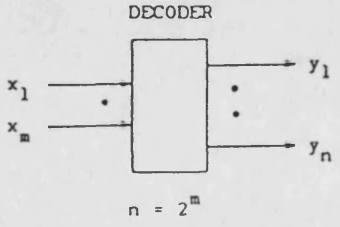


FIGURE 6.31 m TO n LINE DECODER

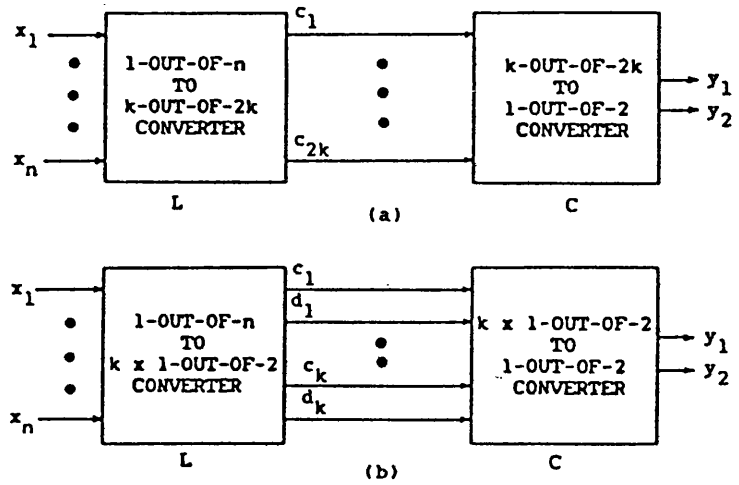


FIGURE 6.32 TSC 1-OUT-OF-n CODE CHECKERS

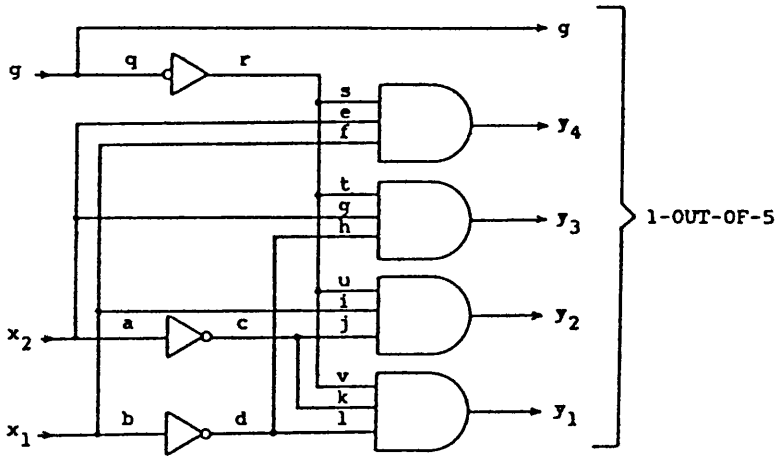
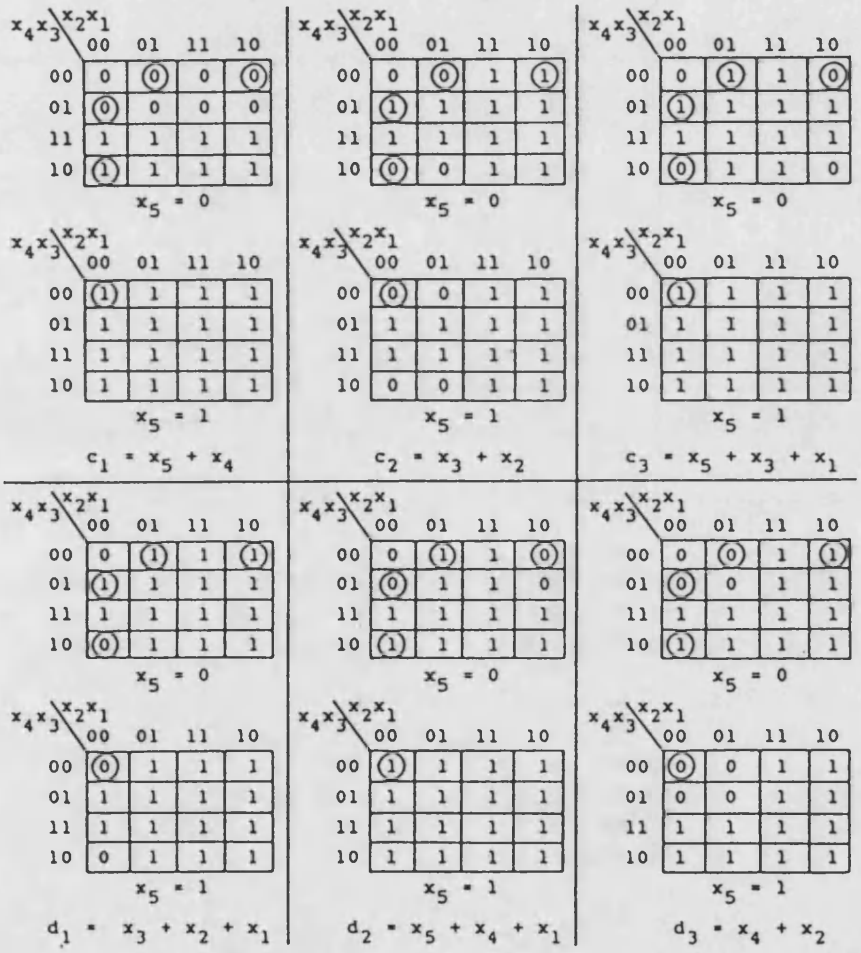


FIGURE 6.33 A 2 TO 4 LINE DECODER WITH ENABLE

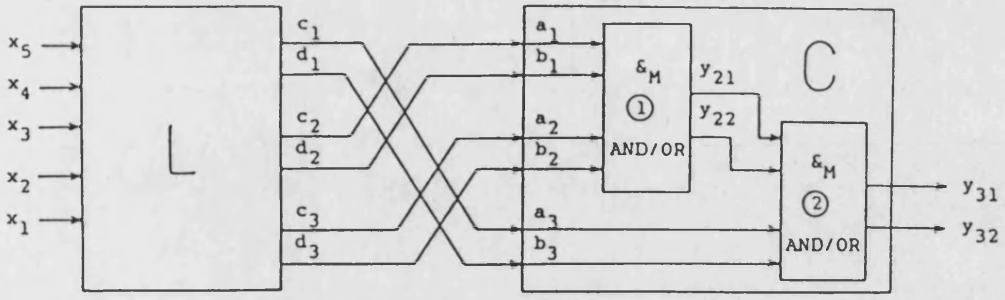
CODE INPUTS					OUTPUTS					
x_5	x_4	x_3	x_2	x_1	c_1	d_1	c_2	d_2	c_3	d_3
0	0	0	0	1	0	1	0	1	1	0
0	0	0	1	0	0	1	1	0	0	1
0	0	1	0	0	0	1	1	0	1	0
0	1	0	0	0	1	0	0	1	0	1
1	0	0	0	0	1	0	0	1	1	0
NOT GENERATED					1	0	1	0	0	1
					1	0	1	0	1	0
					0	1	0	1	0	1

(a) MAPPINGS



(b) KARNAUGH MAPS

FIGURE 6.34 DESIGN OF CIRCUIT L



(a) CIRCUIT

CODE INPUTS								OUTPUTS	
①				②					
a_1	b_1	a_2	b_2	y_{21}	y_{22}	a_3	b_3	y_{31}	y_{32}
0	1	1	0	1	0	0	1	1	0
1	0	0	1	1	0	0	1	1	0
1	0	1	0	0	1	0	1	0	1
0	1	0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0	0	1

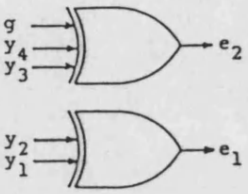
ALL POSSIBLE CODEWORDS

(b) TRUTH TABLE

FIGURE 6.35 DESIGN OF CIRCUIT C

NODE	FAULT	ENABLE g	POSSIBLE EFFECT ON OUTPUT $y_4y_3y_2y_1$
a/b	SA0	0	2 1's
		1	NO EFFECT
	SA1	0	ALL ZERO
		1	NO EFFECT
c/d	SA0	0	ALL ZERO
		1	NO EFFECT
	SA1	0	2 1's
		1	NO EFFECT
e/f/g/h/ i/j/k/l	SA0	0	ALL ZERO
		1	NO EFFECT
	SA1	0	2 1's
		1	NO EFFECT
$y_4/y_3/$ y_2/y_1	SA0	0	ALL ZERO
		1	NO EFFECT
	SA1	0	2 1's
		1	NO EFFECT
q	SA0	0	NO EFFECT
		1	2 1's
	SA1	0	ALL ZERO
		1	NO EFFECT
r/s/t/ u/v	SA0	0	ALL ZERO
		1	NO EFFECT
	SA1	0	NO EFFECT
		1	2 1's

FIGURE 6.36 FAULT ANALYSIS OF FIGURE 6.33



INPUTS					OUTPUTS	
g	y_4	y_3	y_2	y_1	e_2	e_1
0	0	0	0	0	0	0
0	0	0	0	1	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	0	1	1	1
0	0	1	1	0	1	1
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	0	1	1	1
0	1	0	1	0	1	1
0	1	0	1	1	1	0
0	1	1	0	0	0	0
0	1	1	0	1	0	1
0	1	1	1	0	0	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	0	1	1	1
1	0	0	1	0	1	1
1	0	0	1	1	1	0
1	0	1	0	0	0	0
1	0	1	0	1	0	1
1	0	1	1	0	0	1
1	0	1	1	1	0	0
1	1	0	0	0	0	0
1	1	0	0	1	0	1
1	1	0	1	0	0	1
1	1	0	1	1	0	0
1	1	1	0	0	1	0
1	1	1	0	1	1	1
1	1	1	1	0	1	1
1	1	1	1	1	1	0

CODE INPUTS

NON CODE INPUTS WITH CODE OUTPUT

FIGURE 6.37 1-OUT-OF-5 CODE CHECKER

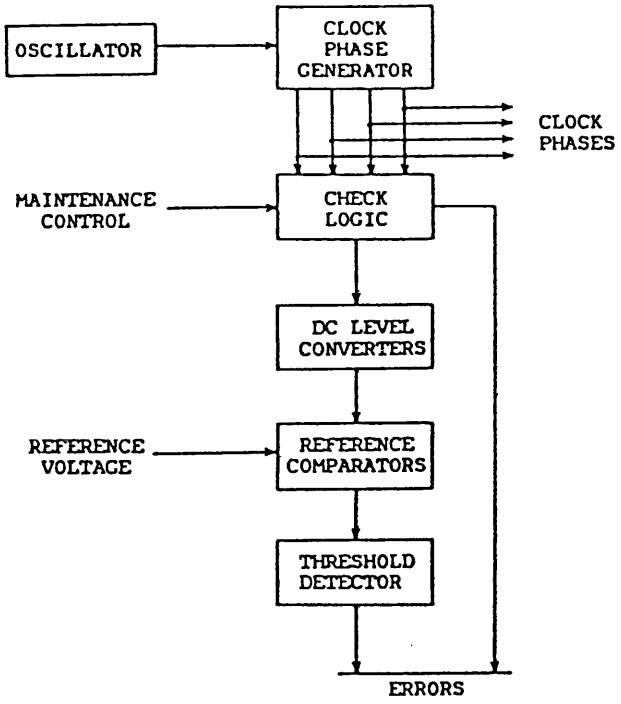


FIGURE 6.38 CLOCK PHASE GENERATION AND CHECKING

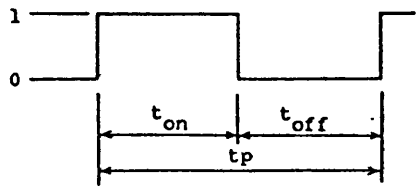


FIGURE 6.39 PERIODIC SIGNAL PARAMETERS

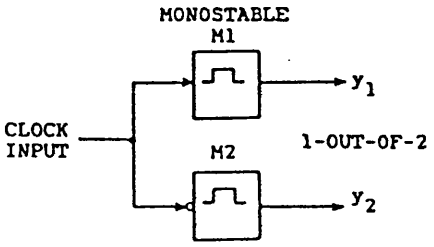


FIGURE 6.40 TSC PERIODIC SIGNAL CHECKER

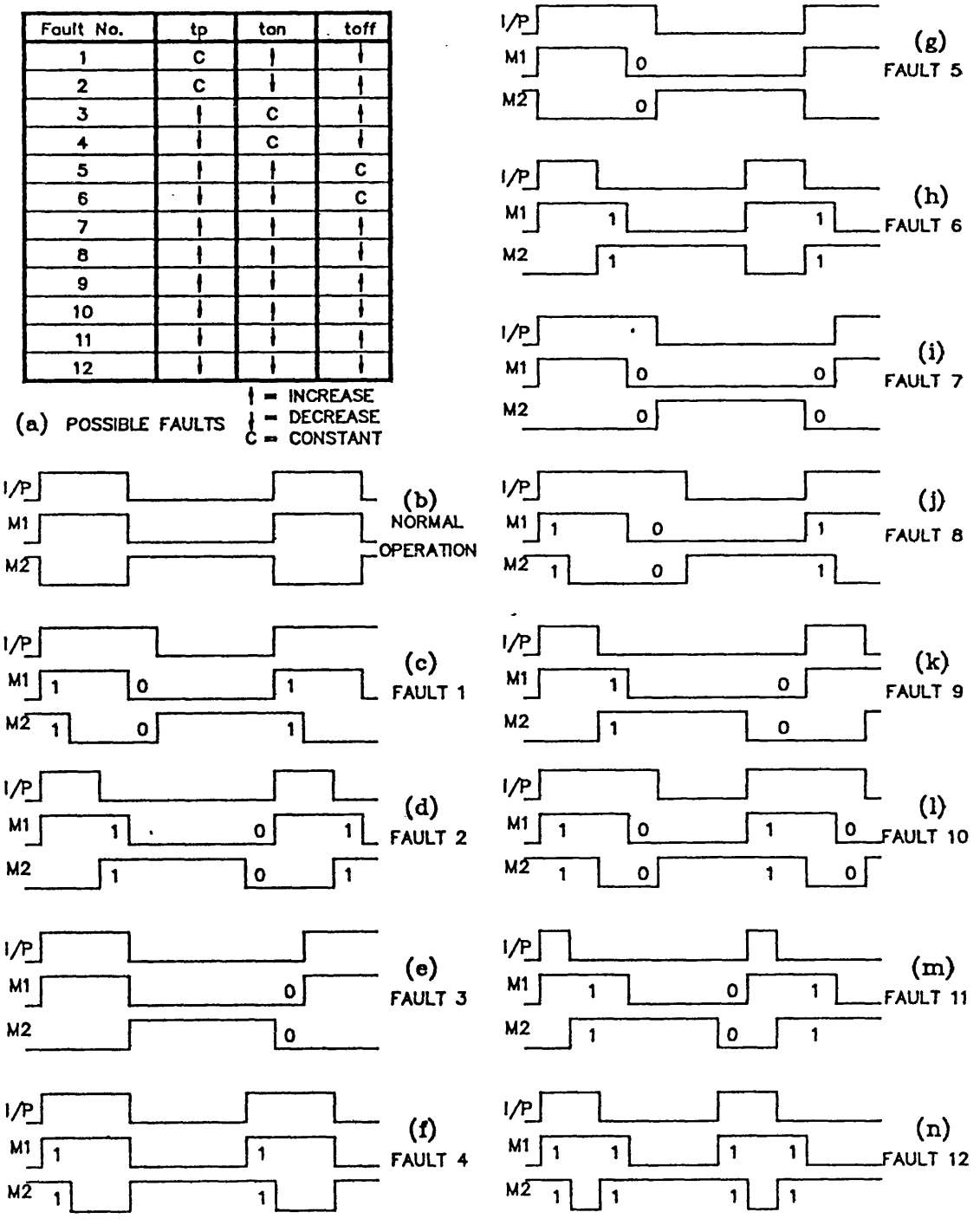


FIGURE 6.41 TSC PERIODIC SIGNAL CHECKER FAULTS AND WAVEFORMS

CHAPTER SEVEN : FAULT DETECTION AND DIAGNOSIS IN SELF CHECKING SYSTEMS

7.1 : INTRODUCTION

The maintenance process for a computer system consists of the following five stages; fault detection, error correction, fault diagnosis/isolation, error recovery and repair. All of these stages are essential in fault tolerant systems, but error correction and error recovery are of particular importance [7.1-7.4,7.34]. Fault detection and fault diagnosis form the basis of this Chapter.

The aim of this investigation is to use self checking techniques as a means of detecting and locating failures at circuit, chip or component level in a microprocessor based system. Whilst error correction and recovery are obvious developments from this aim, they are well documented elsewhere [7.5-7.7,7.35] and not considered here.

There have been many fault tolerant computers developed over the years. Although few of them have used self checking checkers, the self checking philosophies adopted within them are fundamental to systems of the type described in this thesis, which do employ such checkers. Fault tolerant computers in this category include:

1) STAR (Self Test And Repairing) [7.2,7.4,7.8] : One of the earliest and most well known of fault tolerant computers, its development was started in 1961 at the Jet Propulsion Laboratory, under the guidance of Algirdis Avizienis.
2) ESS (Electronic Switching System) [7.5,7.9-7.12] : Various models have been developed at the Bell Laboratories for telephone switching.
3) FTSC (Fault Tolerant Spaceborne Computer) [7.13] : Has a 95% probability of surviving unattended and without degradation for five years.
4) MECRA [7.14] : A self reconfiguring computer developed

in France.

5) Plessey System 250 [7.34] : A fault tolerant modular processing system for control applications.
6) Sperry Univac 1100/60 and System 11 [7.7,7.15] : Both are information processing systems.

The Chapter commences with a discussion on the positioning of self checking checkers within a system. Section 7.3 then examines suitable error detection codes for data transfer paths and memory. Problems caused by selection and addressing errors are also investigated. Various error detection mechanisms are detailed in section 7.4 for the control paths of a system, which include clocks, control lines and decoders. Duplication, check bit correction and check bit prediction are the principal techniques considered for fault detection in arithmetic and logical operations in sections 7.5 and 7.6 respectively. All of the fault detection mechanisms in sections 7.2 to 7.6 are specifically for hardware faults. However, software faults can also occur, so some special hardware checks for these faults are outlined in section 7.7. Finally, fault diagnosis is the subject of section 7.8 which includes fault indication, signal isolation for fault isolation, error indications from transient conditions, the verification and efficiency of fault detection mechanisms and marginal testing.

7.2 : POSITIONING OF CHECKERS

Sellers [7.16] gives criteria for the positioning of checkers in coded data paths as follows:

1) Undetectable errors and faults should be minimised.
2) An error should be detected before it corrupts other data, so that error indications are meaningful for maintenance and so that erroneous data can be reconstructed.
3) The fault should be located to within the smallest amount of hardware necessary to ensure that servicing

is straightforward.

4) The cost of the checker should be minimised.

There must always be trade-offs between the amount and complexity of checkers to satisfy the first three criteria and the minimum cost requirement of the last.

The position and quantity of checkers should therefore enable as many faults and fault types to be detected during normal operation of the system. The measure for such a detection capability is referred to as fault detection coverage. It can be readily determined by examining which component, gate, device or interconnection failures are detected by the fault detection circuits present.

Suitable positions for checkers in a network of self checking blocks are at the outputs of the network and at the inputs of each non code disjoint block. The positioning of checkers must ensure that a network is self testing and fault secure. Individual blocks are assumed to be self testing and fault secure. If the network satisfies theorem 5.1, then a checker is only necessary at its outputs. Network fault security says that all faults will produce the correct output or a non codeword, whilst network self test says that all faults will produce a non code output for some normal network input. If the network contains blocks which are not code disjoint, i.e. those which produce code outputs from non code inputs, then the network is not totally self checking (from theorem 5.1). However, the network can still be fully checked with a checker at its output if additional checkers are positioned at the inputs of these non code disjoint blocks. Wakerley, however, indicates that there need not be a checker on every input [7.17].

The two criteria above give sufficient theoretical positions for the checkers. In practice, additional checkers are positioned to improve the diagnosability of the

network. Sedmak [7.18] gives three categories which should have a high concentration of checkers, particularly if recovery procedures are to be implemented. These are:

1) High failure rate logic.
2) Critical logic and logic with a high usage during normal operation.
3) Storage elements which are accessed infrequently, or where the period between write and read operations is relatively long.

7.3 : DATA PATHS AND MEMORY

7.3.1 : Choice of Error Detection Codes

Chang [7.5] describes the use of a check register to detect multiple errors in data transmission and storage, which does not use an error detecting code and operates as follows. Source data is toggled into an initially all zero check register. After this data is transferred to its destination register, it is immediately read back and toggled into the check register once more. The check register should then again contain all zeros. If not, an error is signalled. Time redundancy is used to perform the additional read operation.

Errors in the transmission and storage of binary data, instructions, and addresses can be detected in data which is encoded in an error detecting code. Codes used for this application are generally separable. They are chosen on the basis of their cost, their effectiveness in detecting the set of likely failures and their impact on system performance. The implementation cost of a code can be determined by the number of redundant bits in the code and by the cost of the appropriate self checking checker. The effectiveness of the code can be principally determined by its minimum Hamming distance (see section 4.3.10), but knowledge of the sizes of its self checking tested and fault secure sets are also significant.

The logical design of data path circuits also plays an important part in determining which faults can be easily detected and require a minimum amount of error checking hardware. Flip-flops, for example, which can be used to form registers for data processing, need to be carefully designed so that logic faults do not result in indeterminate (transient) errors which make diagnosis difficult or impossible. Kraft & Toy [7.19] show a flip-flop design which has a race condition when certain faults are present. They gives two possible solutions to this problem; firstly, additional control lines for diagnostic purposes and, secondly, a circuit redesign to avoid this condition.

Thus the logic structure as well as the error detection code must be considered to achieve the most efficient circuit design from the viewpoints of both error detection and fault diagnosis. The effects of all possible faults must be considered and the error detection code chosen to detect all faults, or at least a reasonable subset.

The circuits used for data paths and memory are generally bit(byte)-sliced circuits with no interaction between bits (bytes), except for control lines (data selection and addressing), which are considered separately in section 7.3.2. If data is encoded in an error detecting code and the circuit is non transforming, i.e. performing only data storage, transmission or switching (see section 5.3.1), then it is self checking, since the output is always a codeword with no failures present. The size of the tested and fault secure sets for such circuits depend on the code used and it is these sets which can be used to compare the effectiveness of each code in detecting faults. Fig. 7.1 details this and other important properties for the separable codes discussed in this Chapter. It is assumed that the normal input set of vectors test for all faults affecting a single bit(byte)-slice, with the relative checker speeds and costs based on the use of SSI and MSI

TTL. For the non transforming circuits of examples 5.1 and 5.2, this means that all input combinations are applied to each bit(byte)-slice during normal operation.

The simplest error dectecting code is single bit parity, discussed in section 4.3.1. This code has n data bits and n+1 bits overall. A non transforming circuit for parity encoded data is fault secure for all faults affecting only a single bit-slice and self testing for all faults affecting fewer than all bit-slices, for reasons given in section 5.3.1. The circuit is thus fault secure for the smallest reasonable fault set, but self testing for a large fault set that contains almost all faults. Single bit parity has minimum redundancy and out of all codes with n data bits, the simplest totally self checking checker. From section 6.7, this checker is an p-bit EXOR tree and an q-bit EXOR tree, such that:

$$p + q = n + 1, \text{ where } p \text{ and } q \geqslant 1 \qquad (7.1)$$

A duplicated circuit is both self testing and fault secure for all faults that affect different combinations of bits in the two circuits. Duplication is therefore better than simple parity, in terms of fault security, but there are many faults which are never tested during normal operation. In particular, any double fault that affects a single bit-slice and its duplicate in the same way is undetectable. Wakerley [7.17] suggests that such a fault is more likely than a fault affecting all n+1 bits in a parity check code. Using this assumption, he shows that parity can be more effective than duplication in terms of its self testing ability. Duplication has the highest redundancy and requires an n-bit TSC equality checker, both of which can involve high costs. It is the most complete and efficient check for certain operations, as there are no constraints with the circuit for fault dectection. Duplication is used where fault types cannot be predicted and where the circuit is not bit(byte)-sliced.

A b-adjacent code, discussed in section 4.3.2, is capable of detecting all errors within a single b-bit byte and is therefore ideal for a b-bit sliced circuit. From section 4.3.2, the code has k x b-bit data bytes and a single b-bit check byte. Its checker consists of b x p-input EXOR trees and b x q-input EXOR trees, such that:

$$p + q = k + 1, \text{ where } p \text{ and } q \geqslant 1 \qquad (7.2)$$

plus a b-input Morphic AND gate. Since the code detects all single byte errors, a non transforming circuit for data in a b-adjacent code is fault secure for all faults that affect a single byte-slice. Analogous to a single bit parity checked circuit, this circuit is self testing for all faults that affect fewer than all byte-slices, since for each of these faults, there is a normal input which produces a non code output. However, it is not necessarily faults affecting all bits of all bytes which are undetectable. For example, a fault which affects bit i of every byte, such that the output from these bits is a codeword, is also undetectable. Overall, though, for applications where faults affecting all bytes of an encoded word are unlikely, the b-adjacent code has the fastest and cheapest checker out of all the codes with b check bits discussed in this Chapter.

The low cost residue codes with check base $A=2^b-1$, discussed in section 4.3.6, are also capable of detecting all errors within a single b-bit byte. Like the b-adjacent codes, they have b redundant bits, but a more costly checker, which employs a tree of b-bit adders. They do not, however, detect errors of magnitude 2^b-1, so a byte which changes from 2^b-1 (all 1's) to 0 (all 0's), or vice versa, is not a detectable error. As a result, a non transforming circuit which employs a low cost code is fault secure for all single byte faults, except all bits stuck at either 0 or 1. It is also fault secure for a large class of multiple byte faults producing

unidirectional errors [7.20]. The circuit is self testing for all faults that affect fewer than all bits, assuming there is no fault in a byte-slice whose only effect is to change 2^b-1 to 0, or vice versa. Such a circuit, when compared to a similar circuit using a b-adjacent code, is thus self testing for a larger fault set, fault secure for a slightly smaller set, has the same redundancy, but its checker is more costly and slower. The low cost residue codes can also be directly used to perform arithmetic operations.

Pradhan and Stiffler [7.21] discuss coding for LSI circuits and transient faults, with the conclusion that the most common faults are unidirectional. They also suggest that Berger codes will be the most important code for the detection of unidirectional errors in future computers, because they are separable codes. In addition, Pradhan has proposed a new code which is not only compatible with parity check codes and able to correct a limited number of errors, but is also able to detect certain unidirectional errors [7.21,7.22]. Error correction is desirable to cope with and recover from the effects of transient errors. Traditionally, the memory unit of a computer has tended to be the most unreliable unit, so error correcting codes are also to be found there.

7.3.2 : Selection and Addressing Errors

The discussion so far applies to the detection of errors in data transmission and storage. The detection of errors in the selection mechanisms for data transfer, or the addressing of data in memory, also need to be considered.

Separate error detection schemes can be used for data and control. Parity on data detects all single bit errors within a register. It is assumed that all faults within the register will only affect one bit. Parity is checked on a data transfer, but does not guarantee that the data has been written to or read from the correct register.

Bit errors during the write operation are detected when the data is read. This is adequate because erroneous data within a register is not a problem until it is read. Decoder faults affecting a complete word may not be not detected, so the control line outputs from the decoder need to be checked, see section 7.4.3.

Decoder faults manifiest themselves, in general, as multiple bit errors in a data word. A decoder check will eliminate the need to detect these errors. However, if the error detection code capability of the data is expanded to include the entire data word, a decoder check is not required.

A simple address decoder for m encoded data words of k-bits is shown in Fig. 7.2. A decoder translates the input address data into a 1-out-of-n code to store, or retrieve the desired word. These words are encoded in an error detecting code such that a storage failure or bus failure is detectable. A decoder failure, however, may result in the selection of the wrong word. The erroneously selected word is a codeword, so the fault is not detected by checking its encoding.

Selection failures can be detected by dividing the memory into bit-slices and providing individual address decoding for each bit, as shown in Fig. 7.3. Now a failure in the selection circuitry produces only a single bit error in the word, which is detected by the error detection scheme for the data.

A single parity check bit will then detect all faults in a single bit slice and its decoder, and single bit faults in the data transfer. The circuit is fault secure for these faults. This technique is inherent in memory chips which are 1-bit wide. If the memory chips are b-bits wide and the error detecting code is capable of detecting b-bit errors, then the decoder circuitry only needs to be duplicated for each byte-slice. Suitable codes are the

b-adjacent (interleaved parity) checksum or Berger codes. For example, if the memory is 4-bits wide, the 4-adjacent code can be used whereby every parity bit covers one bit from each 4-bit slice. This is illustrated in Fig. 7.4. In all cases, the error detection code requires the memory to be one chip wider than normal.

It is assumed in all of the above discussion that if decoder input errors are to be detected, then the address information will also be encoded in an appropriate error detecting code and checked. The technique applied here for decoder circuitry is equally applicable to all other common parts of a memory system, such as driver circuits, refresh logic and bus interfaces. The checking of control lines is futher discussed in section 7.4.

The above scheme takes advantage of data encryption in an error detecting code. It is also possible to detect selection and addressing errors independently of the data error detection code. If, for example, selection failures cause the selection of a data word whose address is different from the desired address by an odd Hamming distance, then these failures can be detected if the parity of the address is stored with each word. The address parity bit is then compared with the actual address parity, when the word is retrieved from memory. The disadvantage of this scheme is that is requires an extra bit to be stored with each word and that it only detects the class of errors described above. A circuit using this scheme is therefore not fault secure for failures that cause a double selection, two words ORed together, or selection of an erroneous word whose address is an even Hamming distance from the address of the desired word. This scheme has been used to detect addressing errors in microprogram control memory [7.5].

Parity can be replaced with any other separable error detecting code. The choice of code is determined by the type of addressing errors to be detected. Other codes

though, in general, will require more than one extra bit to be stored with each data word.

7.3.3 : Code Translation

Whilst it may be desirable to provide byte error detection for the buses of a microprocessor based system, the redundancy of these codes may not make their use in main memory or mass storage economically viable. A code with less redundancy (fewer check bits) may be more desirable for memory. This then requires encoded data to be translated from the memory code to the bus code when it is read from memory and back again when it is written to memory. Totally self checking (TSC) networks for performing this translation are now considered, where both codes are separable.

The interface between buses and memory can be accomplished by the TSC circuitry shown in Fig. 7.5. Data is checked in each direction by a TSC checker for the source code. Check bits are generated for the destination code from the data bits in each case. This generation circuitry is self checking because a generator fault produces either correct check bits, or incorrect check bits making the output a non codeword.

The scheme of Fig. 7.5 requires two check bit generators and two TSC checkers. Each checker can be implemented using a check bit generator and a TSC equality checker, as described in section 5.4.2. As two check bit generators are already required elsewhere, the generators required for the checkers can be replaced with the additional control logic shown in Fig. 7.6. Multiplexers are used to connect the inputs of each check bit generator to the appropriate bus, dependant on the direction of data transfer. This scheme will not be cost effective unless multiplexers are cheaper than check bit generators. This in turn depends on the size of the data bus and the complexity of the check bit generator for the particular

code used.

In Fig. 7.7, the scheme of Fig. 7.6 is modified to provide code translation for data on a bi-directional bus. Check bit tri-state drivers are required instead of data bit multiplexers, which should be cheaper.

Wakerley [7.17] suggests that additional savings can be made for certain combinations of bus and memory codes, where the memory check bits can be directly derived from the bus check bits.

Code conversion could also take place again for arithmetic and logical operation, where, for example, residue codes are more efficient for arithmetic operations than parity. However, the conversion logic must not be over complex, or add an excessive propagation delay to data transfers.

7.3.4 : Interleaved Coding

It is often desirable to retain unmodifed binary information in data tranfers, so that subsequent processing of the data can be performed without any code conversions. In this case, a separable code must be used, the simplest of which is parity. Single parity will not detect a fault which causes an even number of bits to be in error within a data word, but it is still the simplest and most widely used check scheme. However, if multiple bit errors affect adjacent bits instead of being randomly dispersed through the encoded word, as Cook et al suggest [7.11], then these faults can be detected by interleaving parity encoded data information and m-out-of-n encoded control information as in Fig. 7.8. Any multiple adjacent bit fault will affect both the binary data field and the m-out-of-n codeword in the control field. Consequently a single bit parity check is adequate to detect single bit errors errors in the data field, since multiple adjacent bit errors will be detected by the m-out-of-n code check. This interleaved coding technique does not require any additional hardware to give

enhanced error detection.

7.4 : CONTROL PATHS

Previous sections have examined methods of detecting errors in data transfer, storage and manipulation in a computer. There must be some form of control unit to manage these operations. In some instances it may be necessary to duplicate the control unit for error detection purposes. However, in general, there are certain aspects of control which do not require duplication. This section investigates error detection for a number of these aspects.

7.4.1 : Clocks

Errors can occur in the single or multiphase clocking signals which control the various stages of an operation, both within the CPU and I/O devices and between the components of a microprocessor based system (CPU, I/O, memory and discrete logic). Possible clock errors are period and mark-space variation, phase overlap and stuck-at faults. The methods used by Chang for checking multiphase clock signals have already been described in section 6.9. Alternatively, a single phase clock may be used and checked with the TSC periodic signal checker, also described in section 6.9. If multiple phases are required from this arrangement, they can be derived from a faster single phase clock in conjunction with a small self checking sequential circuit.

7.4.2 : Control Lines

Control lines can be grouped together and encoded in an error detecting code just like data lines. The choice of code will be dependent on the level of fault detection required versus the redundancy and cost of the checking scheme. Whilst this method checks the transmission of control signals, it does not guarantee that they reach

their destination correctly. For example, suppose one of the control lines is used to gate bus A on to bus T, as shown in Fig. 7.9. Data on the buses is assumed to be encoded in an error detection code. A break in the 'T ← A' control line affects multiple bits on the T bus and may not be detected by the data path error detecting code. A break at point a in Fig 7.9 will be detected by some input combination on bus A, since it does not affect all bits of the T bus. However, a break at point b can produce an all 0 output for some logic families and will never be detected if all 0's is a codeword in the data path error detection code.

Errors caused by broken lines might seem improbable when considering interconnection wires or PCB tracks, but bad connections in card connectors and integrated circuit sockets, plus wire breaks between bonding pads and pins in integrated circuit packages are all possible faults. A means of detecting errors caused by these failures should therefore be available.

Fig. 7.10 shows a solution to this problem, whereby the control lines are routed to their various destinations and then regrouped for checking purposes at the end of their transmission paths, i.e. after all fan-out points. The checker monitors the data bits of the control word at this point and the check bits at their termination point. Now a failure in the control line(s) is detected by this means, whilst a failure in the same line(s) to an individual bit-slice is detected by the data path error detection code, as shown in Fig. 7.10. If the data path code is byte error detecting, then one line may be tapped off the main control line for each byte-slice.

In the PSC networks of Figs. 5.20-5.23, an inverter may be used to generate s_2 from s_1. A failure in this inverter will be detected by the checking mechanisms for the network, whilst a failure in the main control line (affecting both s_2 and s_1) will be detected by the control

line checking mechanism outlined above.

7.4.3 : Decoders

The example in the previous section used a single control line to provide a particular operation. In practice, there may be many of these lines generated internally, or externally to an integrated circuit for data selection or addressing. If only one line is active at any time, then collectively the lines form a 1-out-of-n code. It is likely that they will have been derived from a $\lceil \log_2 n \rceil$-bit word, as in the case of address decoding for a micro-processor based system, see Fig. 7.11.

Decoder faults are mostly unidirectional. The errors, as indicated in Fig. 6.36 for the circuit of Fig. 6.33, take two forms. Either the selected output changes from a 1 (active) to a 0 (inactive) and there is no active output supplied, or one or more unselected outputs change from a 0 to a 1, giving a multiple output error.

Consider the case of address decoding within a read/write memory chip. When reading data from the chip, decoder failures can cause the contents of two memory locations to be gated on to the data bus, usually producing the logical OR of the two words. This is a unidirectional error. The circuit will be self testing for this type of failure, since, in general, the logical OR of two codewords is a non codeword. However, it is not fault secure since an erroneous codeword could be produced. When writing data to a particular location, failures in the address decoder can cause data to be stored in two locations, with erroneous data replacing the data in one of them. This error may never be detected (when reading the data) because the new data is a valid codeword. In addition, decoder failures could result in no access to any memory location for both read and write operations. If the memory outputs all 1's (or all 0's) for this failure (a unidirectional error) and all 1's (or all 0's) is a codeword, then the failure is

undetectable. Some means other than encoded data must therefore be provided for the detection of decoder failures.

A straightforward check for decoder errors is to duplicate the decoder and compare the outputs of the two circuits. If a fault exists in either decoder, it will be detected by the comparator. The major disadvantage of such a scheme, (aside from identical output faults in both circuits) is that a comparator with a large number of inputs could be required. The alternative means of checking for decoder errors is a 1-out-of-n code checker.

The design of two TSC 1-out-of-n code checkers has already been described in section 6.8. A self testing only 1-out-of-n code checker is detailed in section 5.6. Both Wakerley [7.23] and Kraft & Toy [7.19] present further designs for self tesing only 1-out-of-n code checkers. These compare input data to the decoder with re-encoded outputs from the decoder. Wakerley also shows that this scheme can become TSC if the inputs to the decoder are encoded in a k-out-of-2k code. TSC checkers for m-out-of-n codes in general have also been proposed, see section 6.10 for references.

Comments made in section 7.3.2, on the detection of data selection errors, also apply here. However, the technique of decoder replication may be more expensive than the self testing only and TSC checking schemes discussed above. Decoder replication is practical if a bit or byte-sliced register bank, multiplexer, demultiplexer or functional unit is implemented using conventional MSI or LSI chips which have on-chip decoders.

7.5 : ARITHMETIC OPERATIONS

The basic arithmetic operations are addition, subtraction and shifting. Iterative operations, such as multiplication and division, are accomplished in software or firm-

ware by repeated use of these basic operations. However, as the scale of integration continues to increase and the cost of hardware continue to decrease, hardware multiplier and divider packages become more attractive for systems which require high performance operations. This section briefly discusses error detection for the basic operations using error detection codes. Errors in the iterative operations manifest themselves as errors in the basic operations, or as control errors. Totally self checking checkers are used in all cases.

Typically, binary adders are implemented with either ripple carries or with look-ahead carries, or both. The choice of implementation is governed by the cost of associated hardware and the speed of the arithmetic operation required. The error characteristics of a binary adder depend heavily on its structure. The fault detection scheme must therefore take into account the resultant error caused by logic faults within the adder.

There are several error detecting codes which can be used to directly implement addition and subtraction. The AN codes, as described in section 4.3.5, represent one of these. Low cost AN codes have check bases of the form $A=2^b-1$. In this case, the modulo 2^b-1 addition is performed using ordinary binary adders with end around carry. This is standard one's complement addition. Avizienis [7.24] gives a set of algorithms for multiplication and division of AN encoded data.

The low cost residue codes can also be used for direct implementation of addition and subtraction, with the added advantage of being separate. Wakerley [7.25] gives algorithms for arithmetic operations on low cost residue encoded data. In many of these operations, the check bits must be adjusted in order to produce the correct codeword. This is achieved by adding (or subtracting) bits, derived from either the data or carry parts of the result, to the check bits. The circuitry which controls the check bit

correction must also be checked. Examples of residue checked adders are given by Kraft and Toy [7.19].

In some cases the specialised circuitry required for check bit correction and the propagation delay it imposes, has resulted in designers choosing duplication for a checked ALU design in favour of this technique [7.26]. Duplication is definitely a better choice if the ALU is fabricated on one chip as opposed to byte-slices.

Non arithmetic codes are not preserved by arithmetic operations; i.e. an arithmetic operation performed on two codewords in general produces a non codeword. However, if a separable code is employed, only the check bits resulting from the sum of two codewords are incorrect. New check bits can be generated from the data part of the sum. Check bits generated in this manner, though, cannot indicate errors within the addition operation, because they are derived from the result alone (correct check bits will be generated from an erroneous result). If, however, new check bits are generated from the check bits of both inputs and the resultant data bits, then errors in the addition process will produce errors in the generated check bits. This method is called check bit prediction, previously mentioned in sections 4.3.1 and 4.3.4. The amount of circuitry required to generate new check bits after a logical or arithmetic operation is dependent on the error detecting code used. Wakerley [7.25] presents techniques for the application of check bit prediction with parity and checksum codes. In both cases, carry generation circuitry must be duplicated to detect faults within it. He also examines carry look-ahead structures. Kraft and Toy [7.19] also detail techniques for parity prediction in ripple carry adders and look-ahead carry adders, again stressing the need for duplicated carry circuits.

Binary counting is one of the most important operations within a processor. Its program counter is the most

obvious example. There may also be a requirement for counters in specific application circuitry. In the design of self checking processors or systems, these counters are often checked by parity prediction, whereby the parity of the next count is calculated and subsequently compared with the actual parity for that count. Kraft and Toy [7.19] describe parity prediction for counters in some detail. Self checking counters as well as shift registers are also discussed by Pradhan [7.27].

7.6 : LOGICAL OPERATIONS

Parity codes are preserved by the EXOR operation and a TSC network for computing the EXOR of two k-bit parity encoded words consists of k x 2-input EXOR gates, as Fig. 7.12 taken from reference [7.28] shows.

The duplication code is also preserved by the EXOR function, as well as all other logical functions. A totally self checking logical network can be designed by simply duplicating the functional unit. The output code space of such a network is the duplication code $S = \{\langle DD' \rangle | D=D'\}$, shown in Fig. 7.13. The network is self testing and fault secure for all multiple faults that affect disjoint sets of output in the two logic units. A fault which has the same effect on both logic units is undetectable. The network is neither self testing or fault secure for these faults. In addition to the two logic units, the TSC network has a TSC equality checker, as shown in Fig. 7.13. This indicates an error when the outputs from the two logic units are different.

The outputs from the duplicated logic unit in Fig. 7.13 go no further than the equality checker, for the purposes of the TSC network. If encoded outputs are to be maintained, these outputs could be extended. However, a bit (or byte) error detection code, such as parity, may be adequate for the data transmission paths (see section 7.3). In this case, a TSC code translation will be

required to convert the duplication code to say a parity code, as indicated in Fig. 7.14. The network of Fig. 7.14 is a totally self checking check bit prediction scheme. A TSC code translator can be formed by using the unmodified outputs of one logic unit to supply the output data bits and using the outputs of the duplicated unit to feed a check bit generator, which in turn supplies the output check bits, as shown in Fig. 7.15.

All the above schemes discard the check bits of the input codewords. They are, therefore, not code disjoint, since a non codeword input will produce a codeword output. Checkers are thus required at the logic unit inputs, so that input errors can be detected. Alternatively, duplicated logic units which are code disjoint must be designed. Wakerley [7.29] does this for AND and OR operations using parity, residue and checksum codes.

Wakerley [7.29] also discusses a method of pre-checking logical operations using any error detection code. In this technique errors occurring during a logical operation are not detected, but faults that could affect the operation are tested immediately before it takes place. The disadvantage of this method is that it increases the time to perform such operations (time redundancy).

A development from the concept of pre-checking is that of partially self checking (PSC) circuits and networks, discussed in section 5.5. The PSC logic unit described in section 5.5.4 preserves even parity n-bit input vectors for a number of its functions. If the normal input set to the unit contains a set of code preserving operations which test all single faults in each bit-slice (see section 5.5.4), then the circuit is TSC for this set. Thus all faults can be tested by using a certain set of functions. The unit is not self checking for faults in the control lines before they fan out to individual slices, nor for single slice faults such as input diode shorts, that affect control lines everywhere within the

slice. However these faults can be readily detected by the control line checking techniques discussed in section 7.4. The circuit of Fig. 5.24a can be used in the design of a PSC universal logic unit. The checker will be enabled for code preserving operations and disabled for non code preserving operations, such as AND and OR. Kraft and Toy [7.19] also examine the circuit of Fig. 5.24a, concluding that $A \oplus B$ and its complement $\overline{A \oplus B}$ are capable of exercising every node. Consequently, the parity check for each Boolean logic function needs to be performed on only the EXOR function and its complement.

If certain logical operations are checked, other logical operations can be performed using these checked operations in conjunction with checked arithmetic operations and a set of identities. For example, if the AND operation is checked, other logical operations can be performed using identities such as [7.29];

$$\begin{aligned} A \text{ OR } B &\equiv A \text{ plus } B \text{ minus } A \text{ AND } B \\ A \text{ EXOR } B &\equiv A \text{ plus } B \text{ minus } 2(A \text{ AND } B) \\ \text{NOT } A &\equiv \text{minus } A \text{ (1's complement)} \\ \text{NOT } A &\equiv \text{minus } A \text{ minus 1 (2's complement)} \end{aligned} \qquad (7.3)$$

In this case, the AND operation could be checked by duplication and the arithmetic operations checked by residue codes. Another possibility is to use the parity check code, so that the EXOR operation can be performed by a simple TSC circuit (Fig. 7.12), with parity check bit prediction for the arithmetic operations. Other logical operations can then be computed using an identity such as [7.29]:

$$A \text{ AND } B \equiv 1/2(A \text{ plus } B \text{ minus } (A \oplus B)) \qquad (7.4)$$

7.7 : HARDWARE CHECKS FOR SOFTWARE FAULTS

Aside from hardware faults, software faults can also cause errors in a system. However, software failures may be

more difficult to handle, particularly when little used branches are executed, or when data relevant to a program is corrupted. Hardware checking procedures, as described in previous sections, are designed specifically to detect physical failures in the hardware. They will not, in general, be capable of detecting software faults. Duplicate CPUs will not help either, because a software fault will be executed in the same manner by both processors. In real time applications it is desirable to recognise software faults rapidly and take the necessary corrective action to remove them from the system. Three techniques to detect software faults are as follows:

1) Watchdog Timer.
2) Branch Allowed Check Bit.
3) Different Parity For Instructions and Data.

A watchdog timer is a hardware timer which runs continuously. It is periodically reset by the main program. If, due to a hardware or software fault, the timer is not reset, it times out and appropriate action is taken, such as a high priority interrupt to the processor.

A branch allowed check bit is a means of detecting the execution of an improper branch operation. A check bit, called a branch allowed (BA) bit, is assigned to each word in memory. If the BA bit contains a 0, for example, the contents of that location may not be referenced by any branch instruction. This is illustrated in Fig. 7.16. The bit is checked during every branch operation, with a branch to a word where BA=0 generating an error.

If a parity check bit is assigned to each word in memory, the parity bit can be used to distinguish between instruction and data words. This can be effected by assigning odd parity, for example, to an instruction and even parity to a data word, as shown in Fig. 7.17. A data word referenced as an instruction, due a software or hardware faults, is readily identified and appropriate action taken.

7.8 : FAULT DIAGNOSIS

Fault detection determines whether or not a circuit is behaving correctly. Fault diagnosis identifies the failure to a replaceable unit. This replaceable unit may be a component, circuit, board or subsystem. Traditionally, a fault diagnosis routine uses fault detection hardware and test sequences to locate a defective unit.

If the units are designed in a self checking manner, hardware check circuits detect a fault and identify the defective unit down to the lowest level desired. The level of diagnosis is dependent on the quantity and position of checkers. If duplication and comparison is used for circuits and devices, then a mismatch indicates a failure, but the fault cannot be diagnosed to the functional unit or its duplicate. Thus, both will need to be replaced on detection of an error.

A replacement at board level may be justified on the basis of a faster repair time for the faulty unit, or the cost savings of not having to design additional fault diagnosis hardware and software. However, when a board is replaced, the integrity of the replacement unit must be verified to establish that it does not also possess a fault. It is, therefore, necessary for the complete system to be extensively tested. However, if the self checking circuitry is designed to diagnose faults down to the smallest level possible (component), then this will be unnecessary, as the system will indicate if the replaceent device or board is defective in use. It may take quite some time, though, to generate all vectors in normal operation so that each unit and each checker are self tested, so particular test programs may be run which are specifically designed to exercise the system for this purpose.

In the checking schemes of sections 7.3 to 7.6, errors are

detected concurrently with normal system operation. It is assumed that this error detection occurs before the fault contaminates other parts of the system, making diagnosis difficult. The amount of time which elapses between the occurrence of a fault and its detection is dependent on the position and quantity of checkers, plus the level of fault detection provided by each individual technique adopted.

7.8.1 : Fault Indication

The 1-out-of-2 outputs from each checking circuit (equality, 1-out-of-n and periodic signal checkers etc.) can be combined using an n-input Morphic AND gate to produce a single 1-out-of-2 pair, which indicates an error free or failed system. This signal can then be used to halt or interrupt the processor, store all error signals for subsequent analysis, restart the processor and re-initialise or reconfigure the system.

Error signals from each fault detection circuit provide detailed diagnostic information about the system. This information has to be acquired from the system. This can be achieved by the error signals setting flip-flops in a register (with appropriate test mechanisms if it is not self testing). The contents of this register can be accessed externally and/or displayed on a series of indicator lamps, such as light emitting diodes (LEDs). A fault table can be generated to locate a fault to the desired level from a given set of error indications. This assumes that the fault detection circuit has the ability to detect faults down to the lowest level required.

The error condition in the 1-out-of-2 pair is represented by the vectors ⟨00⟩ or ⟨11⟩, so a single EXOR gate can be used to distinguish error free and faulty vectors, as indicated in Fig. 7.18. However, the EXOR gate is not a self testing circuit, in that certain failures within the gate will mask an error signal from the checker.

(Theorem 6.4 gives the conditions for an EXOR gate to be self testing.) The most obvious failure in this category is the EXOR gate output stuck-at one, indicating a permanent error free condition, despite the outputs of the checker. It is therefore essential that the EXOR gate is checked periodically to ensure that it is operating correctly. An input vector must be applied to verify that it can generate an error indication. The vector should thus be ⟨00⟩ or ⟨11⟩. This can be effected by a special test signal. Fig.7.19a shows one such mechanism, which requires an additional EXOR gate. Under normal conditions the test input is at 0. This allows the ⟨01⟩ and ⟨10⟩ vectors from the check circuit to pass directly to the output EXOR gate, as in Fig. 7.18. The test input set to a 1 primarily confirms the operation of the output EXOR gate, but also checks the additional EXOR gate. Fig. 7.19b gives the fault table for these two gates. Now with ⟨01⟩ or ⟨10⟩ output vectors from the checker, the input vectors to the output EXOR gate are ⟨00⟩ and ⟨11⟩ respectively,which means that this gate should output a 0.

7.8.2 : Fault Isolation

In general, there will be a fan-out from every logic gate output, i.e. every gate output will be connected to at least one gate input. This is illustrated in Fig. 7.20a, where a single transmitter (gate output) sends data to many receivers (gate inputs). Another configuration is possible, shown in Fig. 7.20b, where a single receiver accepts data from one of several transmitters. The transmitter will be either wire ORed (or wire ANDed) together, or have tri-state output buffers. This latter configuration is generally associated with bus structures.

Consider now the encoded and checked bus structures of Figs. 7.21a and 7.21b. For diagnostic purposes the output of each buffer block is checked for failures. The fault tables in Figs. 7.21a and 7.21b identify the faults that are detected by each checker in the circuit. These reveal

that any input or output fault that causes the bus to become stuck at a particular logic level is undiagnosable. For example, in Fig. 7.21a, if the bus becomes stuck-at-0 (SA0), it is unknown from the checker indications if this is caused by the output of buffer B1 SA0, or the inputs of buffers B2 or B3 SA0. The only solution, aside from using a current probe to determine which device is sinking current and therefore faulty [7.30], is to replace each device connected to the bus in turn, until the fault is eliminated. Sedmak [7.6], incidentally, investigates the structure of Fig. 7.21a, but does not consider bus failures caused by gate input failures.

An alternative solution is to introduce signal isolators at strategic points in the signal lines, so that there is only one output or input connected directly to the bus. Then a stuck bus line (assuming that the lines themselves are not open or short circuit) can only be due to the directly connected device. The structures of Fig. 7.21a and 7.21b now become those of Figs. 7.22a and 7.22b respectively. The isolation circuits are allowed to fail, but not in such a mannner as to cause a stuck-at failure where they are connected to the bus. An input to output short is another failure which is not allowed. Either of these failures would render the isolator circuit useless.

The fault tables in Figs. 7.22a and 7.22b now reveal that faults in individual devices are diagnosable to that device from checker indications. The additional checkers (C3 and C4) shown in Fig. 7.22b serve two purposes; firstly they detect isolator faults and secondly they improve the diagnosability of this structure. Isolator failures are modelled as producing identical fault indications as the single input or output of the device they are connected to and isolate. Each isolator and the device it isolates are then considered as one unit and replaced as such. This is an ideal situation, which in practice is dependent on the actual implementation of the isolator circuitry.

Moreira de Souza et al, in their research oriented computer [7.31], have used a series resistor and CMOS buffer as an isolator for each signal line in Fig. 7.22a and a series diode for each signal line in Fig. 7.22b, resulting in Figs. 7.23a and 7.23b respectively. The resistors used are wirewound types, where the probability of an open circuit is much greater than the probability of a short circuit. The CMOS buffers ensure that minimal voltage is dropped across the resistors, so that an acceptable noise immunity is maintained. The tie-up resistors on the transmitter outputs in Fig. 7.23b maintain odd or even parity for the checker between a transmitter and its isolator when that transmitter is inactive. Appendix B shows that both short and open circuit diode failures are detected in this structure.

In the STAR computer [7.4], Avizienis used the arrangement shown in Fig. 7.24 for input and output isolation from a bus. Unfortunately, the series static redundancy employed for the isolation components is useless, since a failure within them cannnot be detected. If, for example, one of the output diodes become a short circuit, then the isolator is still functional, but the failed diode must be detected before the other diode fails as well. Avizienis does not indicate how this is effected, if at all.

It is envisaged that these isolators would, ultimately, become integrated, or at least gate circuitry redesigned so that physical input and output stuck-at failures could not occur. Wirewound resistors are not easily integrated, so an alternative isolator to that in Fig. 7.23a had to be designed for the self checking system of Chapter 9. The circuit adopted is that in Fig. 7.25.

Both Moreira de Souza and Avizienis only consider isolators for unidirectional buses. In Fig. 7.26a, the structures of Fig.7.22 are merged to from the structure of a bidirectional bus. The self checking system of Chapter

9 has a bidirectional data bus, so a bidirectional isolator was required for this purpose. This is shown in Fig. 7.26b. The circuits of Figs. 7.25 and 7.26b are derived in Appendix B.

7.8.3 : Transient Error Indications

The outputs of each checker and/or the combined error signal pair can be monitored continously or loaded into a register at certain defined intervals, for example on an edge of a system clock. If continuously monitored, there may be certain regular transient errors which occur during the time interval when a number of signals are changing state.

An example of this is the decoder in Fig. 6.33. Assume that its inputs change from ⟨11⟩ to ⟨00⟩. However, primarily because of variations in the gate delays encountered by each line, the inputs may go through the following transitions: 11 → 01 → 00. These transitions give the decoder output waveforms shown in Fig. 7.27 and result in a transient spike from output y_2. These waveforms indicate that the decoder generates a 1-out-of-n code output at all times, but the spike may cause a transient error in another part of the system. Assuming they are not a problem, these transient errors will be ignored if the checker outputs are sampled only when the system signals are known to be stable. Alternatively, if checker outputs are monitored continuously, the transient error indications can be eliminated by filtering the checker outputs, to remove transient errors of less than a defined duration, or by preventing the transient spikes from occurring. The latter solution is effected by inhibiting the decoder outputs until the input signals are stable. Alternatively, if the input data to the decoder is encoded, the code can be chosen such that all possible intermediate transitions between codeword inputs are non codewords. In this case output spikes do not occur. M-out-of-n codes are suitable for this purpose [7.19].

7.8.4 : Verification and Efficiency of Fault Detection Circuits

As indicated in the section 7.8.1, any non self testing section of fault detection hardware or subsequent error processing circuitry must be periodically exercised to ensure that it is fully operational. This process uses hardware to simulate test conditions or circuit faults and software to control it. The error detection circuits can be checked automatically by generating an error condition and observing their response. For parity and m-out-of-n code checkers, for example, a non codeword needs to be generated by the system. This can be effected by applying special test signals as stimuli to the check hardware, so that an error is created.

In reference [7.5] Chang describes mechanisms to exercise check circuits. The data transfer and storage parity checkers are exercised by reading idle or power on memory locations, i.e. memory which has not been written to. Hopefully, some of these will contain words with incorrect parity. Several words with incorrect parity are included in the system ROM or EPROM for a similar purpose. A 1-out-of-n decoder checker is exercised by selecting an unused decoder output which, by design, is not an input to the code checker. The checker is therefore given the impression of no output selected. A second test is the selection of another unused output, which in this instance, again by design, is fed twice into the checker. The checker is now given the impression of two active lines.

The Sperry Univac 1100/60 uses a combination of software and hardware to verify that fault detection, isolation and recovery mechanisms are operational [7.7]. This capability is provided by fault injection, a process of deliberately causing a fault to occur within a system by the insertion of erroneous data or control signals into

a portion of logic covered by a fault detection circuit. The need for such testing arises because the section of logic which processes the error signals is not frequently exercised during normal operation and is not self testing.

Fault injection is also the subject of a paper by Crouzet and Decouty [7.32]. In order to determine the effectiveness of the fault detection mechanisms in their research computer , described in section 8.3.10, they inject over 75,000 faults into it. The efficiency of the fault detection mechanisms, defined as the probability of detecting a fault knowing that one has occurred, was over 99%. A measurement of the safety of the detection mechanisms, defined as the probability of the system producing error free output data, knowing that a fault has occurred, was 100%. Both test and application programs where used whilst injecting faults, with the test program giving a higher rate of fault detection, as expected. This type of analysis highlights weak points in the design for self checking. In the case of the research computer, these were found to be multiplexed bus checking and the implementation of Morphic logic functions.

7.8.5 : Marginal Testing

In most of the discussion so far on error detection, it has been assumed that circuit faults were hard faults. These faults are always repeatable, since the same errors will always be generated under the same set of operating conditions. However, a logic gate or circuit as well as failing instantaneously due to, for example, electrical overstress, can also fail gradually, in that its operating voltage and current levels no longer fall within specified tolerance levels over a period of time. This gradual degradation creates maginal operating conditions, which lead to transient errors, as the circuit appears to be functioning correctly for most of the time. In addition, such marginal conditions may cause failures which are not

repeatable under the same set of circuit conditions. These faults are therefore extremely difficult to identify. Kraft & Toy [7.33] consider aspects of marginal testing, whereby voltage threshold levels are adjusted, dynamic memory refresh rates reduced and clock rates to the CPU increased.

7.9 : SUMMARY AND CONCLUSIONS

Section 7.2 has considered the positioning of checkers in a network of self checking blocks. However, whilst these may be the minimum requirements to detect all faults from a specific set, many additional checkers are required in practice, depending on the level of fault diagnosis required. In the experimental computer of Chapter 9, for example, fault diagnosis is desired at chip or circuit level.

Fault detection in data transmission paths is effected with an error detecting code. The level of fault detecion is dependent on the code chosen. The choice of code is also governed by the amount redundancy it invokes and the cost of generating and checking it. If different codes are adopted for data paths and memory, then code translation circuitry will also be required.

Faults in the control circuitry of memory can be detected without the need for a separate check, if either the memory is organised in bit or byte-slices, for bit and byte data error detecting codes respectively, or the check bits of the address error detecting code are stored with the data and checked when the data is retrieved from memory.

If the information on the data transmission paths is parity encoded, then the detection of errors in this information can be improved above that of parity, by interleaving it with encoded information whose code has better error detection capabilities than parity.

Control lines must also be checked. Some control line failures will produce errors in the data transmission paths and be detected indirectly. Other faults will not be detected in this manner, so the control information must be encoded and checked. Decoder outputs, for example, which are inherently 1-out-of-n encoded, must be checked with a totally self checking 1-out-of-n checker. The checking scheme for control lines must always include a checker at the end of their transmission paths, to ensure that all breaks in these lines are detectable.

Circuitry performing arithmetic operations can be checked by duplication and comparison, by encoded inputs where the error detecting code is preserved by the arithmetic operation(s) (with additional circuitry for check bit correction), or by encoded inputs with check bit prediction circuitry. Chapters 8 and 9 indicate that duplication and comparison is the most widely used technique. Despite its redundancy, the technique requires a minimum of design effort to implement.

Circuits performing logical operations can be checked by duplication and comparison, by encoded inputs using a set of code preserving operations which test all single faults in each bit-slice, or by encoded inputs with check bit prediction crcuitry. Alternatively, if certain logical and arithmetic operations are checked, other logical operations can be performed via a set of identities using these checked operations. Again duplication and comparison is the most widely used technique.

Three hardware checks for software faults have been described. These are a watchdog timer, a branch allowed check bit and different parity for instructions and data. However, there appears to be little attention paid to software faults, and therefore these tests, in the self checking systems reviewed in Chapter 8.

The level of fault diagnosis is dependent on the quantity, position and checking ability of the self checking circuits. All 1-out-of-2 outputs from the checkers can be combined to produce a single 1-out-of-2 pair, which will take some action on detection of an error, but all individual outputs are required for diagnostic purposes. The individual signals and/or the combined signal may also be visually displayed. The diagnostic and display processes will involve some non self testing circuitry. Provision must be made to periodically test this circuitry.

Although the fault detection logic is self checking, it can be checked, along with the error processing logic, by special hardware and software provided to create errors in the system. This is a form of fault injection, which can also be effected by using external mechanisms to simulate various faults. The efficiency of the fault detection circuits can also be evaluated using fault injection.

The outputs from each checker may be monitored continuously to observe all transient errors, but unwanted transition spikes will have to be eliminated from the system. In general, however, the checker outputs are sampled at regular defined points during system operation, for the detection of transient and permanent faults.

Mechanisms may also be provided within the system for marginal testing, so that degradations in operating conditions are detected. These mechanisms appear to be rare, however.

Fault diagnosis is assisted by the inclusion of isolating circuits between a gate output (transmitter) and the gate inputs (receivers) it drives. These are ideally integrated as discrete package or within the device they are isolating. Circuits adopted in the experimental self checking computer of Chapter 9 are discussed in

Appendix B.

Various techniques for fault detection and fault diagnosis have been presented in this Chapter. It is important to examine the extent of their usage in current and past research, not only as a means of putting them into perspective, but also to determine how far the aims of this investigation have already been met. A number of self checking devices and systems are therefore reviewed in Chapter 8 for this purpose. Many of the techniques for fault detection and fault diagnosis, presented in this Chapter, are practically applied in the experimental self checking computer of Chapter 9.

7.10 : REFERENCES

7.1) Designing reliable computer systems; the fault tolerant approach - 1 - R. G. Bennetts; IEE Electronics and Power; November/December, 1978; pp. 846-851.

7.2) Designing reliable computer systems; the fault tolerant approach - 2 - R. G. Bennetts; IEE Electronics and Power; January 1979; pp. 51-56.

7.3) Fault tolerance and digital systems - R. G. Bennetts; Microprocessors and Microsystems; Vol. 3, No. 8; October, 1979; pp. 365-373.

7.4) Fault tolerant computing: Techniques and development - A. Avizienis; Infotech*; pp. 309-333.

7.5) Maintenance techniques of a microprogrammed self-checking control complex of an electronic switching system - H. Y. P. Chang, G. W. Heimbigner, D. J. Senese, T. L. Smith; IEEE Trans. Comput., Vol. C-22, No. 5; May, 1973; pp. 501-512.

7.6) Fault-tolerance of a general purpose computer implemented by very large scale integration - R. M. Sedmak, H.L. Libergot; FTCS-8*; pp.137-143.

7.7) Availability, reliability and maintainability aspects of the Sperry Univac 1100/60 - L. A. Boone, H. L. Liebergot, R. M. Sedmak; FTCS-10*; pp. 3-8.

7.8) The STAR (self-testing and repairing) computer: An investigation of the theory and practice of fault-tolerant computer design - A. Avizienis, G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, D. K. Rubin; IEEE Trans. Comput.; Vol. C-20, No. 11; November, 1971; pp. 1312-1321.

7.9) Microprogrammed control and reliable design of small computers - G. D. Kraft, W. N. Toy; Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA; 1981; Chapter 9, pp. 368-415.

7.10) Design of a microprogram control for a processor in an electronic switching system - T. F. Storey; Bell Syst. Tech. J.; February, 1976; pp. 183-232.

7.11) Design of a self-checking microprogram control - R. W. Cook, W. H. Sisson, T. F. Storey, W. N. Toy; IEEE Trans. Comput.; Vol. C-22, No. 3; March, 1973; pp. 255-262.

7.12) The design of a microprogrammed self-checking processor of an electronic switching system - H. Y. P. Chang, R. C. Dorr, D. J. Senese; IEEE Trans. Comput.; Vol. C-22, No. 5; May 1973; pp. 489-499.

7.13) Architectural design for near - 100% fault coverage - J. J. Stiffler; FTCS-6*; pp. 134-137.

7.14) Design techniques for reliable hardware - F. P. Maison; Infotech*; pp. 236-251.

7.15) Installability, availablity, reliability and maintainability aspects of the Sperry System 11 - A. K. Bhatt, D. R. Mueller; FTCS-14*; pp. 29-35.

7.16) Error detecting logic for digital computers - F. F. Sellers, M. Y. Hsiao, L. W. Bearnson; McGraw-Hill, New York; 1968.

7.17) Error detecting codes, self checking circuits and applications - J. F. Wakerly; Elsevier - North Holland; New York; 1978; Chapter 4, pp. 101-122.

7.18) Implementation techniques for self-verification - R. M. Sedmak; 1980 Test Conf.*; pp. 267-278.

7.19) as per 7.9); Chapter 7, pp. 212-330.

7.20) as per 7.17); Chapter 2, pp. 9-53.

7.21) Error-correcting codes and self-checking circuits - D. K. Pradhan, J. J. Stiffler; Computer; March, 1980; pp. 27-37.

7.22) A new class of error-correcting/detecting codes for fault-tolerant computer applications - D. K. Pradhan; IEEE Trans. Comput.; Vol. C-29, No. 6; June, 1980; pp. 471-481.

7.23) as per 7.17); Chapter 7, pp. 156-174.

7.24) Arithmetic algorithms for error-coded operands - A. Avizienis; IEEE Trans. Comput.; Vol. C-22, No. 16; June, 1973, pp. 567-572.

7.25) as per 7.17); Chapter 5, pp. 123-142.

7.26) Cost effectiveness of self checking computer design - W. C. Carter, G. R. Putzola, A. B. Wadia, W. G. Bouricius, D. C. Jessep, E. P. Hsieh, C. J. Tan; FTCS-7*; pp. 117-123.

7.27) Shift Registers designed for on-line fault detection - D. K. Pradhan, M. Y. Hsiao, A. M. Patel, S. Y. Su.; FTCS-8*; pp. 173-178.

7.28) as per 7.17); Chapter 3, pp. 54-99.

7.29) as per 7.17); Chapter 6, pp. 143-155.

7.30) HP 547A Digital Current Tracer; Hewlett Packard 1986 Product Catalogue; Hewlett Packard,USA;p. 378.

7.31) A research oriented microcomputer with built in auto-diagnostics - J. Moreira de Souza, E. Peixoto Paz, C. Landrault; FTCS-6*; pp. 3-8.

7.32) Measurement of fault detection mechanisms efficiency : Results - Y. Crouzet, B. Decouty; FTCS-12*; pp. 373-376.

7.33) as per 7.9); Chapter 8, pp. 331-367.

7.34) Computer systems reliability: An overview - A. Avizienis; Infotech*; 1975; pp. 216-233.

7.35) Fault diagnosis in computer control systems - G. Edge; Systems Technology; No. 31; April, 1979; pp. 33-41.

* see section B.5.

Code	Single bit odd parity
Data Bits	n
Check Bits	1
Fault Security	Single bit-slice faults
Self Test	Faults affecting fewer than all bits
Checker	p-bit EXCLUSIVE OR tree and q-bit EXCLUSIVE OR tree (p+q=n+1, where p and q $\geq$ 1)
Comments	Least redundancy, cheapest checker
Code	Duplication
Data Bits	n
Check Bits	n
Fault Security	Fault affecting different bits in the two circuits
Self Test	Fault affecting different bits in the two circuits
Checker	n-bit TSC equality checker
Comments	Most redundancy, expensive checker, not self testing for many double faults
Code	b-adjacent (odd parity)
Data Bits	k x b-bit bytes
Check Bits	1 x b-bit byte
Fault Security	Single byte-slice faults
Self Test	Faults affecting fewer than all bytes
Checker	b x p-bit EXCLUSIVE OR trees, b X q-bit EXCLUSIVE OR trees and b-bit TSC Morphic AND gate (p+q=k+1, where p and q $\geq$ 1)
Comments	Cheapest checker out of all codes with b check bits
Code	Low cost residue, A = 2^b-1
Data Bits	k x b-bit bytes
Check Bits	1 x b-bit byte
Fault Security	Single byte slice faults except all bits stuck-at-1 or stuck-at-0
Self Test	Faults affecting fewer than all bits
Checker	k-byte tree of b-bit modulo 2^b-1 adders and TSC equality checker
Comments	Direct implementation of arithmetic operations, slower and more expensive checker compared with b-adjacent code checker

FIGURE 7.1 PROPERTIES OF FOUR SEPARABLE CODES

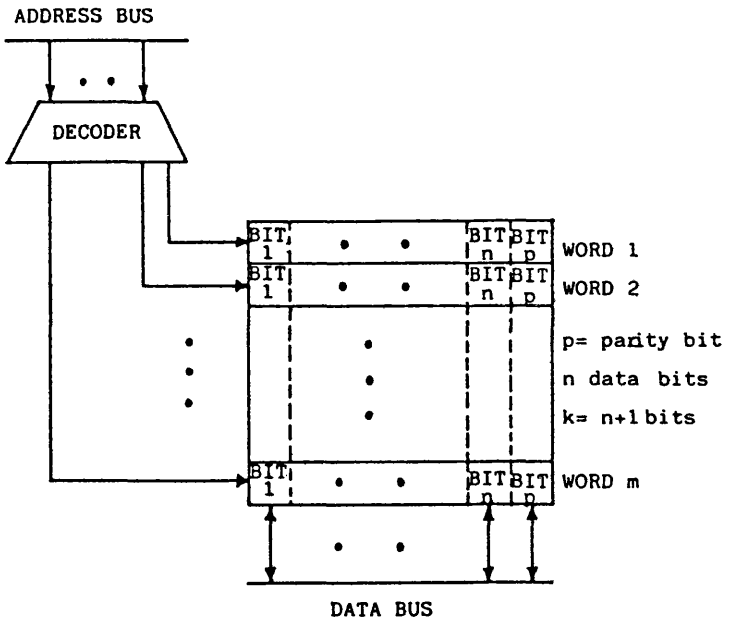


FIGURE 7.2 MEMORY ADDRESS DECODING

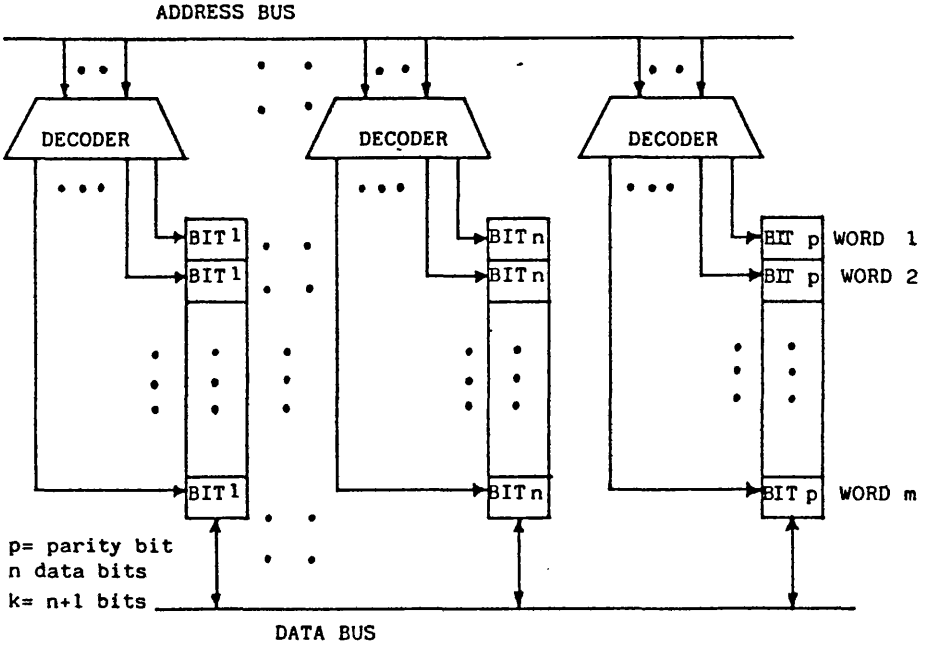


FIGURE 7.3 MEMORY ADDRESS DECODING WITH REPLICATION

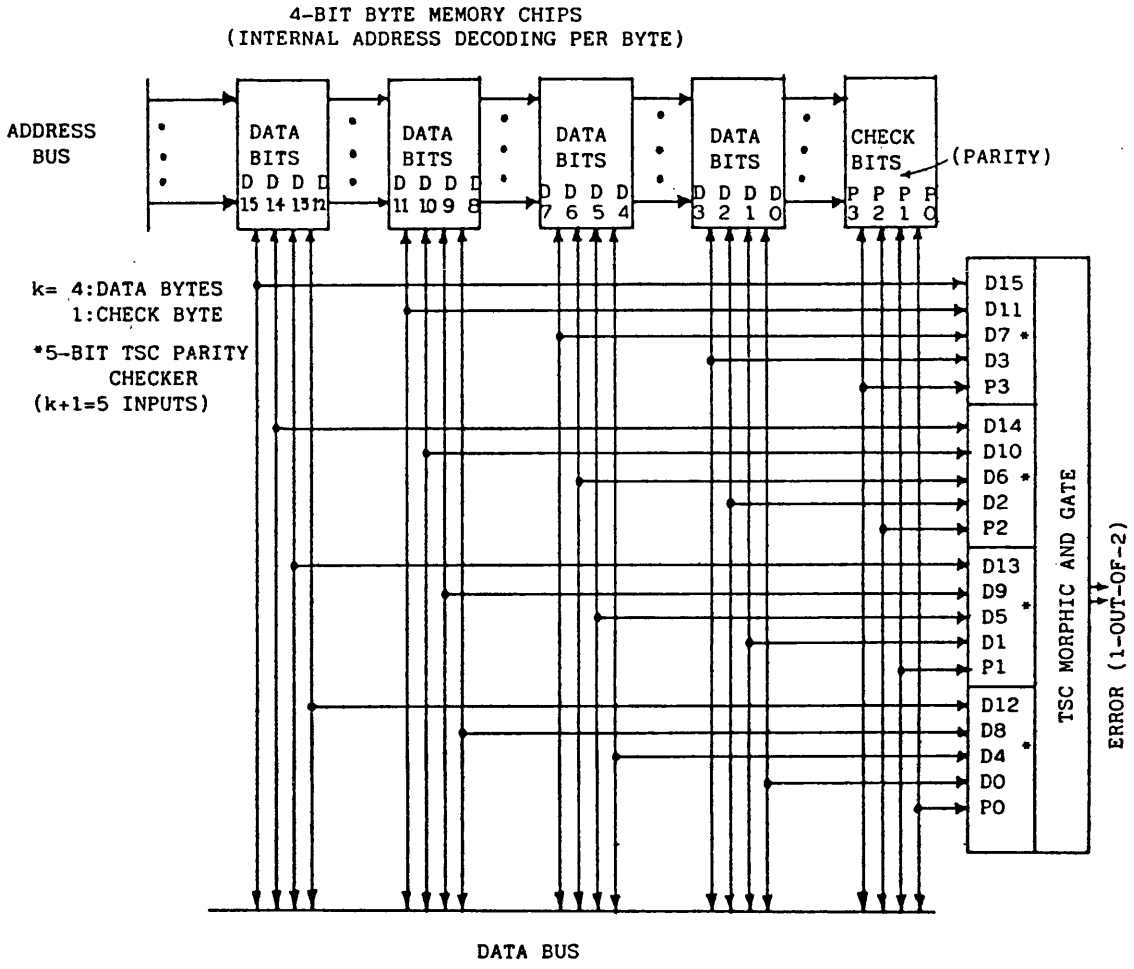


FIGURE 7.4 4-ADJACENT CODE CHECK

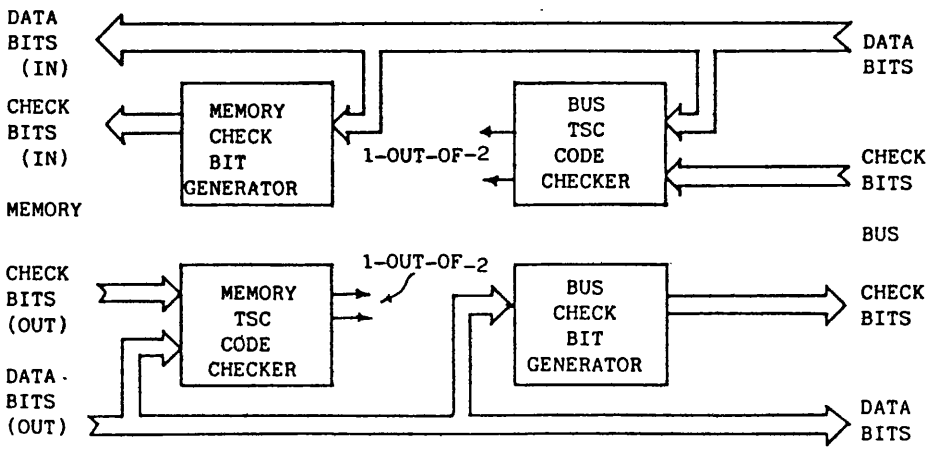


FIGURE 7.5 CODE TRANSLATION FOR UNIDIRECTIONAL BUSES

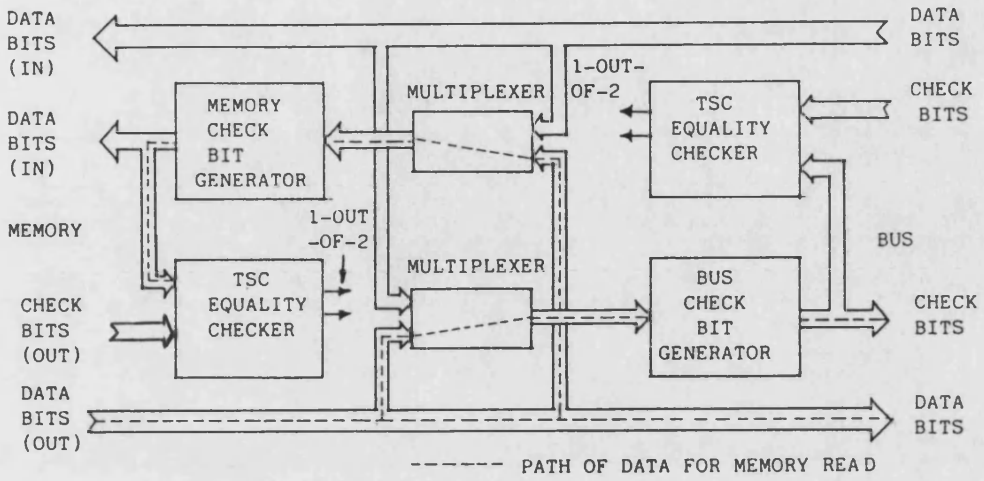


FIGURE 7.6 CODE TRANSLATION FOR UNIDIRECTIONAL BUSES WITH COMMON CHECK BIT GENERATION

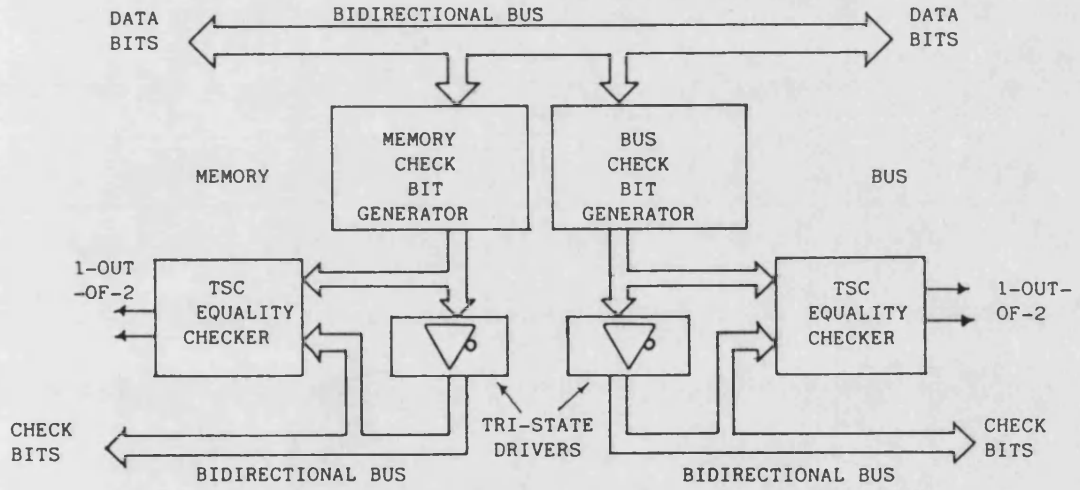


FIGURE 7.7 CODE TRANSLATION FOR BIDIRECTIONAL BUSES

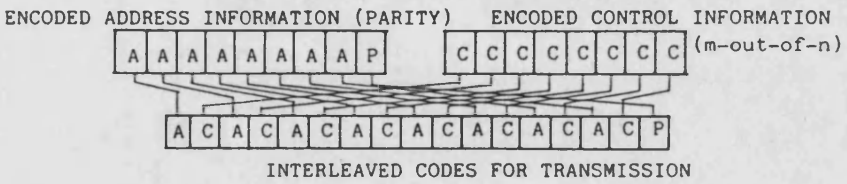


FIGURE 7.8 INTERLEAVED CODES

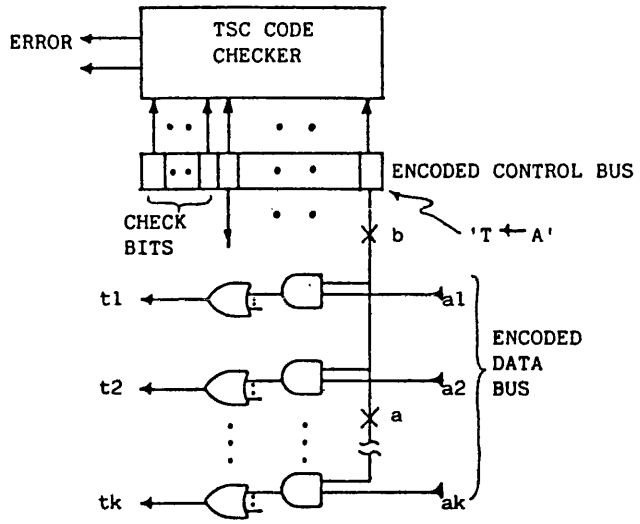


FIGURE 7.9 INEFFECTIVE CONTROL LINE CHECK

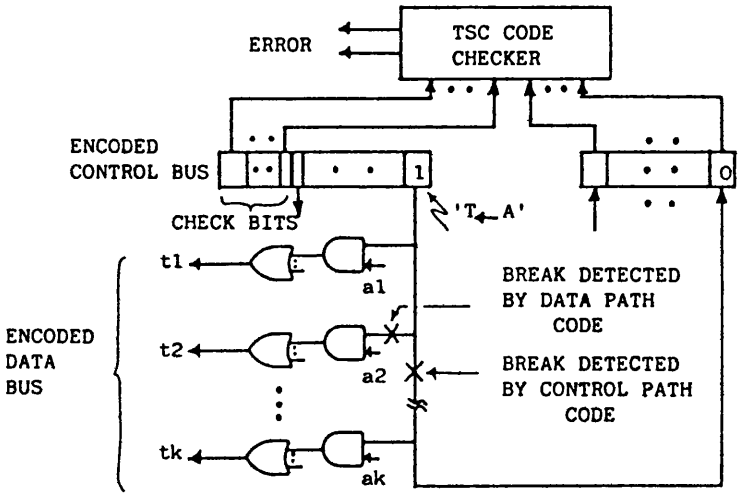


FIGURE 7.10 EFFECTIVE CONTROL LINE CHECK

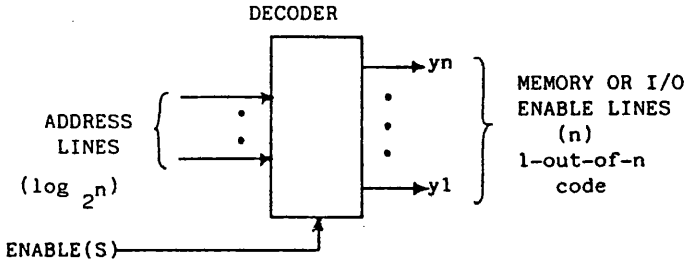


FIGURE 7.11 ADDRESS DECODING

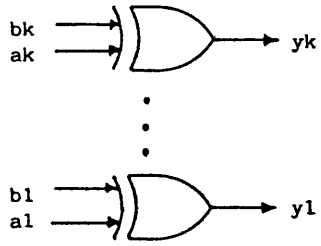


FIGURE 7.12 TSC CIRCUIT FOR THE EXOR OF TWO PARITY ENCODED WORDS

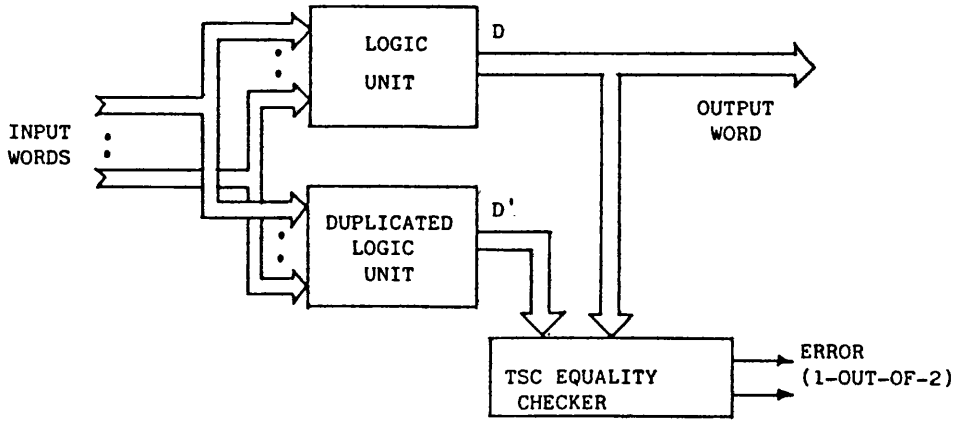


FIGURE 7.13 TSC LOGICAL NETWORK

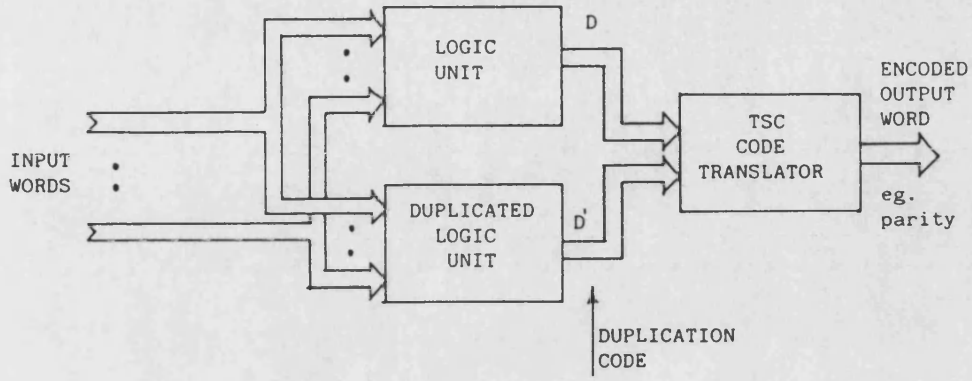


FIGURE 7.14 TSC CHECK BIT PREDICTION

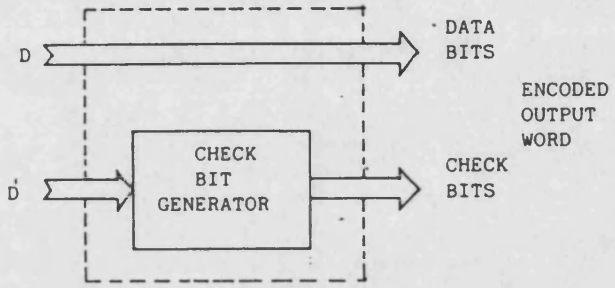


FIGURE 7.15 TSC CODE TRANSLATOR

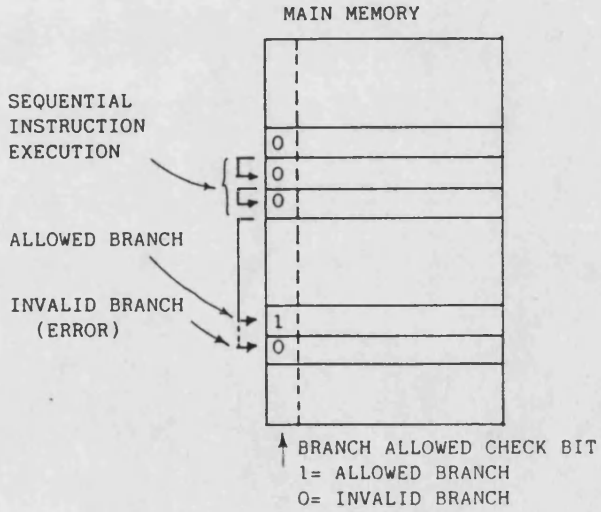


FIGURE 7.16 BRANCH ALLOWED CHECK

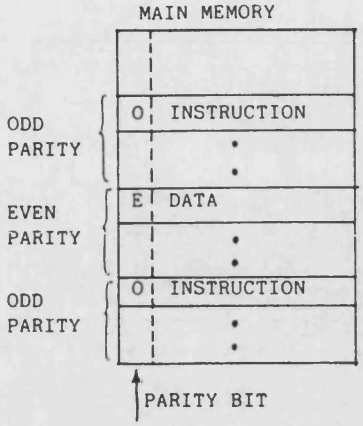


FIGURE 7.17 DIFFERENT PARITY FOR INSTRUCTION AND DATA WORDS

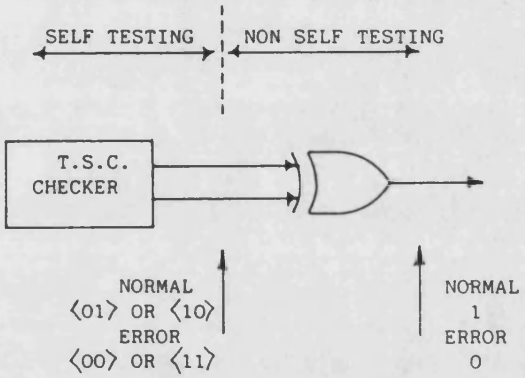
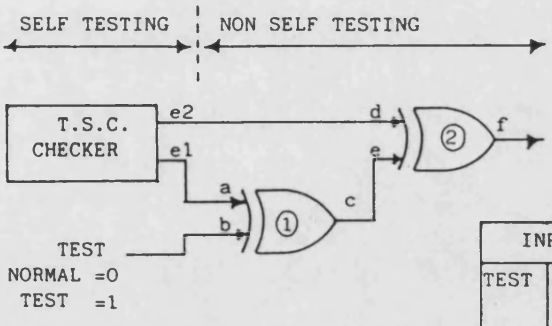


FIGURE 7.18 TWO RAIL (1-OUT-OF-2) TO SINGLE RAIL CONVERSION



(a) CIRCUIT

(b) FAULT TABLE

INPUTS		GATE ①	GATE ②	
TEST	e2 e1 (CODE)	a b c	d e f	
0	0 1	0 1 0	1 0 0	NORMAL MODE
0	1 0	1 1 1	0 1 0	
1	0 1	0 0 1	1 1 1	TEST MODE
1	1 0	1 0 0	0 0 1	

0=SA0 detected
1=SA1 detected

FIGURE 7.19 TESTING THE TWO RAIL TO SINGLE RAIL CONVERTER

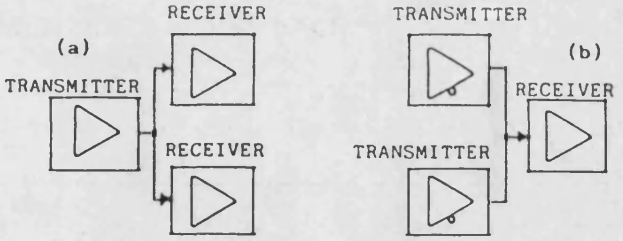
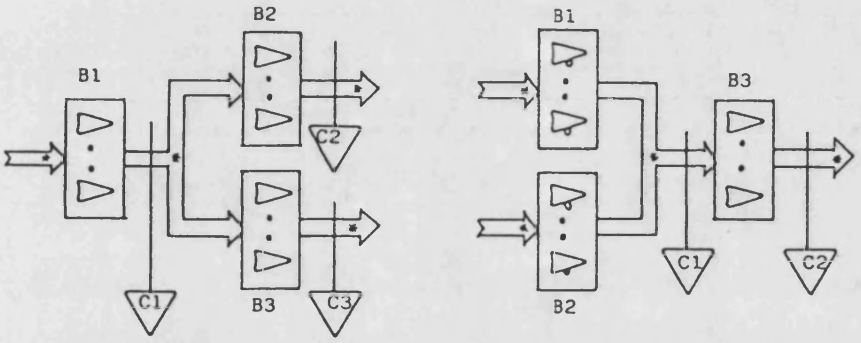


FIGURE 7.20 TRANSMITTER AND RECEIVER CONFIGURATIONS



FAULT	CHECKER(S) DETECTING FAULT		
	C1	C2	C3
B1 O/P	X	X	X
B2 I/P	X	X	X
B2 O/P		X	
B3 I/P	X	X	X
B3 O/P			X

* = CODED BUS
B1-B3 = BUFFER PACKAGES
C1-C3 = TSC CHECKERS
FAULT = SINGLE FAULT IF CODE IS PARITY

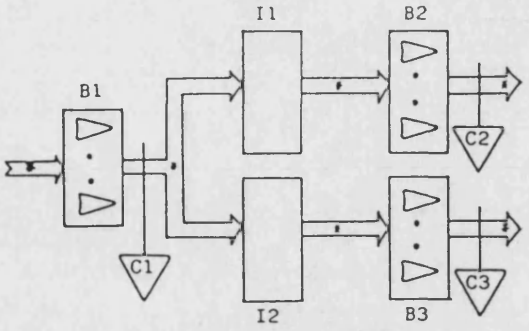
(a)

FAULT	CHECKER(S) DETECTING FAULT			
	B1 TRANSMITS TO B3		B2 TRANSMITS TO B3	
	C1	C2	C1	C2
B1 O/P	X	X	X	X
B2 O/P	X	X	X	X
B3 I/P	X	X	X	X
B3 O/P		X		X

* = CODED BUS
B1,B2 = TRI-STATE BUFFER PACKAGES
B3 = BUFFER PACKAGE
C1,C2 = TSC CHECKERS
FAULT = SINGLE FAULT IF CODE IS PARITY

(b)

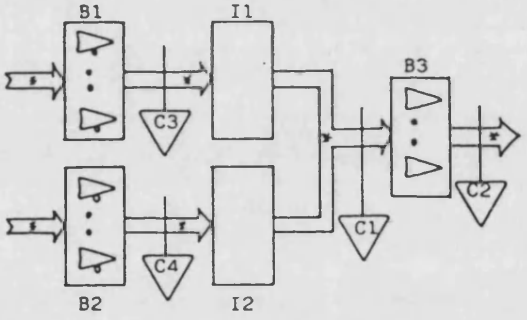
FIGURE 7.21 CHECKED BUS STRUCTURES



FAULT	CHECKER(S) DETECTING FAULT		
	C1	C2	C3
B1 O/P	X	X	X
I1		X	
B2 I/P		X	
B2 O/P		X	
I2			X
B3 I/P			X
B3 O/P			X

* = CODED BUS
B1-B3 = BUFFER PACKAGES
C1-C3 = TSC CHECKERS
I1,I2 = ISOLATOR PACKAGES
FAULT = SINGLE FAULT IF CODE IS PARITY
ISOLATOR FAULT = AS ABOVE, BUT NEVER AN INPUT STUCK-AT FAULT OR INPUT TO OUTPUT SHORT CIRCUIT

(a)



FAULT	CHECKER(S) DETECTING FAULT							
	B1 TX TO B3				B2 TX TO B3			
	C1	C2	C3	C4	C1	C2	C3	C4
B1 O/P	X	X	X				X	
I1	X	X	X				X	
B2 O/P				X	X	X		X
I2				X	X	X		X
B3 I/P	X	X			X	X		
B3 O/P		X				X		

* = CODED BUS
B1,B2 = TRI-STATE BUFFER PACKAGES
B3 = BUFFER PACKAGE
C1-C4 = TSC CHECKERS
I1,I2 = ISOLATOR PACKAGES
FAULT = SINGLE FAULT IF CODE IS PARITY
ISOLATOR FAULT = AS ABOVE, BUT NEVER AN OUTPUT STUCK-AT FAULT OR INPUT TO OUTPUT SHORT CIRCUIT

(b)

FIGURE 7.22 CHECKED BUS STRUCTURES WITH ISOLATORS

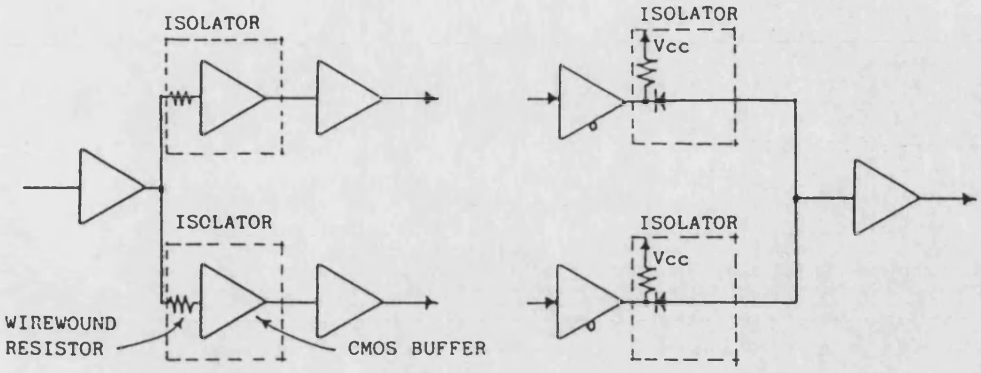


FIGURE 7.23 ISOLATOR CIRCUITS USED BY MOREIRA DE SOUZA

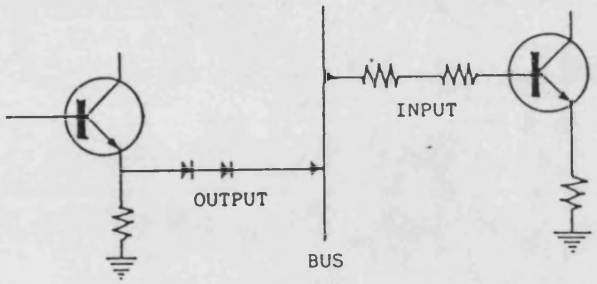


FIGURE 7.24 ISOLATOR CIRCUITS USED BY AVIZIENIS

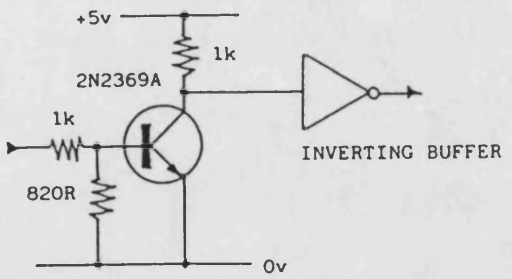
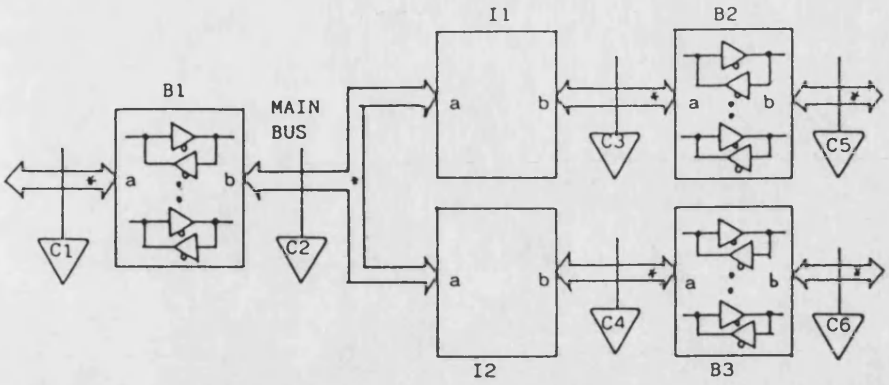


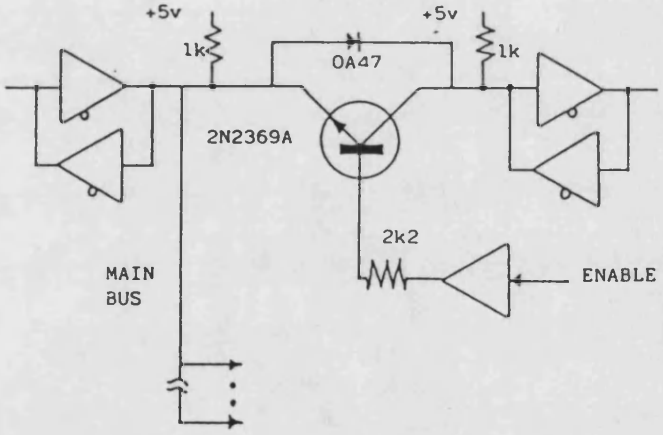
FIGURE 7.25 UNIDIRECTIONAL ISOLATOR USED IN CHAPTER 9



FAULT	CHECKER(S) DETECTING FAULT																							
	B1 TX TO B2						B1 TX TO B3						B2 TX TO B1						B3 TX TO B1					
	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6
B1a	X	X	X		X		X	X		X		X	X						X					
B1b		X	X		X			X		X		X	X	X					X	X				
I1b			X		X								X	X	X									
B2a			X		X								X	X	X									
B2b					X								X	X	X		X							
I2b										X		X							X	X		X		
B3a										X		X							X	X		X		
B3b												X							X	X		X		X

* = CODED BUS
B1-B3 = TRI-STATE BIDIRECTIONAL BUFFER PACKAGES
C1-C6 = TSC CHECKERS
I1,I2 = BIDIRECTIONAL ISOLATOR PACKAGES
FAULT = SINGLE FAULT IF CODE IS PARITY
ISOLATOR FAULT = AS ABOVE, BUT NEVER A STUCK-AT FAULT ON THE MAIN BUS OR AN UNDETECTABLE TERMINAL TO TERMINAL SHORT CIRCUIT

(a) STRUCTURE



(b) Isolator circuit

FIGURE 7.26 BIDIRECTIONAL BUS STRUCTURE AND ISOLATOR

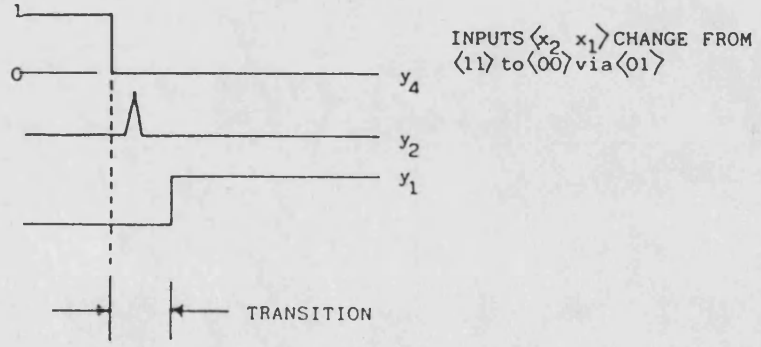


FIGURE 7.27 OUTPUT TIMING OF DECODER IN FIG 6.33

CHAPTER EIGHT : SELF CHECKING SYSTEMS AND DEVICES : A REVIEW

8.1 : INTRODUCTION

One of the reasons that self checking systems are not often designed using commercially available integrated circuits is the high cost involved to do so. Many SSI and MSI devices are required to implement such a system, as the required functions are not available in LSI or VLSI form. Possible solutions to this problem are to develop circuits incorporating self checking (or fault tolerant) mechanisms specfic to the function performed by that circuit, or to develop specific self checking circuits containing a set of mechanisms which allow self checking to be incorporated in a system using traditional circuits.

Some circuits based on the first category are already available. The Motorola 68000 and the Intel iAPX 432 are two examples which have been already discussed in section 4.9. The most widespread fault tolerant device is an interface circuit for Hamming single error correcting/ double error detecting (SEC/DED) coded memory, which is available from several manufacturers.

If self checking techniques are taken into account during the design of LSI/VLSI integrated circuits, this reduces the hardware increase for the introduction of self checking at system level. It also enables designers who are not familiar with self checking techniques to incorporate them into a system.

A number of self checking devices have been proposed for use in self checking systems. However, in general, self checking versions of existing integrated circuits have not been investigated. So, before reviewing a number of proposed self checking devices, section 8.2 examines the modifications required to convert an existing integrated circuit into a self checking form. Sections 8.3.1 to

8.3.8 then briefly review some proposed self checking devices. Devices from both categories mentioned above are included. Following this, sections 8.3.9 to 8.3.11 detail various proposed self checking systems.

8.2 : SELF CHECKING INTEGRATED CIRCUITS

If a totally self checking system is to be constructed out of integrated circuits (ICs) which have a built in self checking ability, then all ICs must be available in a self checking form. There must therefore be self checking NAND gate packages (SSI) through to self checking microprocessors (VLSI). The problem is to select a self checking technique which can be adapted for any application that the IC might be used for.

If, for example, the outputs of a NAND gate package were parity encoded, a parity checker could monitor the normal gate outputs, as well as the additional pin required for parity, to constantly check the output code for failures, as shown in Fig. 8.1. However, as previously discussed for check bit prediction in section 7.5, any failure producing an erroneous gate output within the IC will not be detected. This is because the parity generation is based on gate outputs. The only solution is to implement the complete parity prediction technique by encoding the gate inputs as well. This approach is certainly appropriate for VLSI devices, such as microprocessors, which have a bus architecture. It is also the approach adopted by Sedmak for his generalised self checking VLSI chip, described in section 8.3.7.

Alternatively, and perhaps more appropriate for SSI packages, is a duplication and comparison of internal circuitry in normal or complementary form (see section 8.3.7). It is more appropriate, because each gate is used, in general, for a completely independent purpose. The NAND package shown in Fig. 8.1 now becomes that shown in Fig. 8.2. Two additional pins are required for each IC

to output the 1-out-of-2 encoded error signal pair. Signal pairs from each IC could drive individual fault indicators (via latches), be merged in groups to drive fewer indicators, or be merged to give a single fault indication. The merging process would use n-input Morphic AND gates, then available in IC form (see section 8.3.8). However, inputs faults, such as stuck at lines due to short or open circuits, right up to the input line fan-out points within the IC (see Fig. 8.2) would not be detected. Erroneous input signals would be fed to both copies of the circuit. This problem would be minimised if the fan-out point of each input was at its IC pin. Inputs could be encoded and checked at this pin, if this was desired, but error signals from each IC would be a true indication of the operational state of the IC itself. An additional problem to be avoided would be an internal fault which caused the error signals to become stuck at ‹01› or ‹10›.

8.3 : PROPOSED SELF CHECKING SYSTEMS AND DEVICES

8.3.1 : 68000 Microprocessor

The Motorola 68000 microprocessor has already been discussed in section 4.9.2 from the view point of its built in test capabilities. Nicolaidis has evaluated a self checking design of this processor [8.1]. He pays particular attention to the designs of other researchers for a self checking sequencer, indicating that in each case there are some real failures within the circuit which are not detected. In suggesting a satisfactory sequencer he discusses aspects of self checking ROMs, PLAs and arithmetic units.

8.3.2 : MIL-STD-1750A Microprocessor

One of the designs studied by Nicolaidis is the VLSI self checking MIL-STD-1750A microprocessor proposed by Halbert and Bose [8.2]. They make extensive use of PLAs to form TSC building blocks, which are then interconnected to form

larger TSC circuits. Using this means they discuss two major microprocessor subsystems, the arithmetic unit and the microprogram controller. In both cases the objective is 100% fault coverage.

8.3.3 : Microprogram Control Unit

Wong et al have designed a self checking microprogram control unit [8.3] based on the Advanced Micro Devices AM2910 microprogram controller [8.4]. The effect of physical failures on the outputs of each functional block are checked, rather than each individual physical failure. Fault assumptions are arbitrary failures in functional units and unidirectional errors on control or data transfer buses and stored words. On this basis, Berger, modified Berger codes and duplication codes are used. The modified Berger code requires fewer check bits than a normal Berger code. It is used to check n-bit data words in which the number of ones is known to be always much less than n (up to 3 in this instance). The unit uses only three totally self checking checkers, all of which are equality checkers.

8.3.4 : Four-Bit Microprocessor

Crouzet et al [8.5,8.6] consider the attributes of five error detection codes; duplication, k-out-of-2k, Berger, b-adjacent and residue. Using the constraints of MOS circuit design, they detail the coding and checking circuits required for each code in terms of the number of gates (power levels), the number of transistors (size) and the number of gate levels (time delay). This forms a basis for code comparison. They then apply parity (single error detection), duplication (multiple error detection) and k-out-of-2k encodings to a specific 4-bit microprocessor. A graphical representation of the processor unit is used to apply each of the three coding techniques in turn, with particular attention to the positioning of the checkers. They conclude that the areas of the processor which employ

duplication require an increase in hardware which is of the same order as the increase required by either of the other two. Overall duplication is favoured because it is easier to design and will detect a large set of faults.

8.3.5 : Error Detection Processor

Chavade et al [8.7,8.8] propose a self checking error detection processor for use with commercially available integrated circuits in the design of a self checking system, as depicted in Fig. 8.3. The facilities of the processor include:

1) Data comparison.
2) Code translation and checking (translation between parity, 4-adjacent or SEC/DED Hamming codes and duplication, or vice versa).
3) Watchdog timer (hardware/software errors).
4) Failure management (error data logging and recovery).

It is able to cope with duplicate synchronous or asynchronous processors and duplicate or encoded blocks devoted to data transfer or storage. It is programmable to select the operating mode, cascadable from the point of view of fault signals and is ideally housed within a 40-pin package.

8.3.6 : The PAD

The PAD is a development of the error detection processor by the same team [8.9]. It is again a self checking LSI device for fault detection in a microcomputer and is used with standard integrated circuits to form a self checking system, as shown in Fig. 8.3.

The system must have twin CPUs but the memory and I/O blocks can be either duplicated or encoded. For duplicated parts the PAD provides:

1) Comparison.
2) Data exchange between CPUs.
3) CPU sychronisation.

For non duplicated parts it provides:

1) Generation and checking of three codes (single parity, 4-adjacent and SEC/DED Hamming codes).
2) Control of non duplicated peripherals.

In addition to these architecturally dependent functions, the PAD also caters for the following:

1) A watch dog timer check.
2) Detection of memory protection violations.
3) Error management (transient error recovery).

The device allows CPU synchronisation to the nearest clock cycle (micro synchronised), or CPU synchronisation at program segment or task level (macro synchronised). All the PAD fault detection mechanisms are implemented in self checking logic. The phases adopted for designing the self checking logic are described, along with the influence of its integration.

8.3.7 : Generalised VLSI Chip

Sedmak has proposed a generalised self checking VLSI chip, shown in Fig. 8.4, which has the following features [8.10-8.12]:

1) Functional logic which is duplicated in complementary form.
2) N comparators to check the two functional block outputs as well as intermediate results.
3) Redundant power inputs checked by comparison.
4) Output data and control information encoded in parity or an error correcting code.
5) M input data and control line code checkers.

6) Error encoding and multiplexing logic to generate encoded output error signals from the internal checker outputs.
7) 4-phase input clock check.

Complementary logic is used for the duplicated circuitry, because it eliminates the occurrence of identical mask, cell or bridging faults in the two circuits. The signals in and out of the complementary circuit are opposite to those of its duplicate. The same physical devices are used for sequential logic, but control signals and stored data are opposite in polarity to those in the duplicate circuit. An example of a circuit and its complement is given in Fig. 8.5. Sedmak also proposes an error handling chip to process the outputs from each individual self checking VLSI chip. This chip, shown in Fig. 8.6, merges and sorts the information to isolate a fault.

Overall this scheme results in:

1) Immediate detection of all single, most multiple and most bridging faults.
2) Immediate detection of power and clock failures.
3) An opportunity to recover from these failures.
4) Automatic isolation of the failed chip or interchip connections.
5) A chip count increase of 5.5% over a conventionally checked VLSI design.

8.3.8 : Morphic AND Gate

In reference [8.13] Wakerley proposes a 4-input Morhic AND gate with polarity control to allow input encodings in either the duplication or 1-out-of-2 codes. The circuit is to be housed in a 14-pin package containing 4 x 2-input gates, 16 x 4-input gates and two 8-input gates, as shown in Fig. 8.7. This is the design given by equation (6.9) and shown in Fig. 6.23b. Five of these chips would be required for a 16-bit equality checker. One advantage of

using SSI gates is that they can be organised to be well checked for single failures. The custom chip approach is quite vulnerable to failures in which a pair of lines on the chip become stuck at ‹10› or ‹01›. This can be avoided by using a separate custom chip to generate each output line of the pair (independent subcircuits), at the expense of increasing the number of chips required. However, assuming only unidirectional failures, the custom chip approach is quite adequate.

8.3.9 : VLSI Building Blocks

Rennels et al have presented the results of a study to establish a standard set of VLSI building blocks [8.14]. These blocks are assembled with commercially available microprocessors and memory into fault tolerant distributed computer configurations. The resulting multi-computer architecture uses self checking computer modules. A redundant bus system is employed for communication purposes between modules. The blocks use many of the techniques discussed in this Chapter. They are designed to meet a number of important conditions as follows:

1) They must interface directly with a variety of commercial microprocessors and memory.
2) Existing bus and I/O standards must be closely followed.
3) The resulting system architecture must have a high fault coverage.

The self checking computer module consists of four building blocks.

1) An error detecting and correcting memory interface which provides Hamming correction to damaged data, replacement of a faulty bit with a spare, parity encoding and decoding of internal buses and detection of internal faults.
2) A Programmable Interface which can be programmed to

perform either the function of a bus adapter or a bus controller.

3) The Core which detects CPU faults by synchronously comparing two CPUs, collects fault indications from itself and other building blocks, attempts a recovery from transient faults and disables the processor on detection of a permanent fault.
4) Digital I/O to provide eight functions, which include serial and parallel I/O with appropriate checking techniques.

8.3.10 : Research Microcomputer

In the real time reasearch oriented microcomputer with built in auto diagnostics, Moreira de Souza et al place a great emphasis on safety, by preventing the transmission of erroneous data to its outputs [8.15]. Using commercially available components, they achieve this with the following:

1) Duplication and comparison of the CPU (Intel 8080).
2) Parity encoding of bit-sliced memory data and I/O data.
3) Duplication and comparison of control circuitry.
4) A fault tolerant clock [8.16] with inhibit switches.
5) Bus isolation (described in section 7.8.2).
6) Fault Analysis Module.

The comparators, parity checkers and fault analysis module are all implemented as self checking circuits. The 1-out-of-2 encoded error signals from the various checkers are processed by the fault analysis module, which indicates the failed module and inhibits the clock. A system with 8 I/O ports, 4K words of RAM and 4K words of ROM requires 2.8 times the amount of hardware as a system without built in autodiagnostics, but this figure decreases with increased memory and I/O. However, it is suggested that if special checking circuits were available, this figure could be reduced to a 1.15 increase.

8.3.11 : Self Checking Computer Costs

As a significant development from their early work, Carter et al have studied the cost effectiveness of a self checking computer design [8.17]. The design is completely self testing. It also has a retry capability without speed degradation and significant additional hardware. The processor unit consists of ten chip types. The particular checking technique employed for each chip is described, along with the additional hardware overheads that this checking creates for each chip, in terms of the number of additional gates required within the chip, the number of additional chip pins required and the number of additional chips.

They also propose a check chip which can perform any one of the following functions:

1) Compute and check even byte parity on two bytes producing 2 parity signals and one checkable 1-out-of-2 pair.
2) Compute the parity of 27 lines producing a single output.
3) Act as a 12-input Morphic AND gate, or as a 11-input Morphic AND gate and check 4 odd parity lines, or compare 2 pairs of lines, producing one checkable 1-out-of-2 pair.

Thirteen of these check chips are used within the self checking computer. Overall, the additional hardware required for checking purposes is again dependent on the amount of memory, but for 8K words it is 38%. The system has byte-sliced storage and is designed to detect all single faults. This results in considerable multiple fault coverage, typically 64-80%.

8.4 : CONCLUSIONS

The self checking devices and systems reviewed above show that duplication and comparison, along with coding, are the most widely used techniques for self checking purposes. Programmable logic arrays (PLAs) are often used to implement the self checking mechansims.

Coding is principally applied to data transmission paths (control, address and data information) and memory. Duplication and comparison is employed for more complex circuitry, such as control and arithmetic units.

Parity continues to be the most predominant form of coding used, because of its minimum redundancy and because it requires a minimum amount of circuitry for generation and checking purposes. However, its error detection capabilities are becoming a major limitation, particlarly in VLSI circuits, so more complex codes, such as Berger codes, are beginning to be adopted. Control information is often encoded in an m-out-of-n code.

Duplication and comparison is extensively employed because it is so straightforward to implement. The technique requires a minimum amount of design effort and will also detect a large set of faults. Even if a complete module is not duplicated, it is likely that the more complex parts of it will be, simply because there is no other viable technique. The duplicate circuitry can be constructed in normal or complementary form, depending on whether it is separate from, or integrated within the functional circuitry.

The main aim of this investigation is to develop a microprocessor based system with fault diagnostics provided by self checking circuits. Existing research most relevant to this aim is the research oriented computer of Moreira de Souza et al [8.15]. However, whilst discussing the various self checking techniques adopted in their system, they do not discuss fault detection, fault indication and error control logic in any detail. There is also no

comment on fault diagnosis for their system. In addition, the research computer uses unidirectional data buses, so there are no checking and isolation considerations for bidirectional buses.

All of the above aspects are detailed in the description of an experimental self checking computer, which follows in Chapter 9.

8.5 : REFERENCES

8.1) Evaluation of a self-checking version of the MC68000 microprocessor - M. Nicolaidis; FTCS-15*; pp. 350-356.

8.2) Design approach for a VLSI self-checking MIL-STD-1750A microprocessor - M. P. Halbert, S. M. Bose; FTCS-14*; pp. 254-259.

8.3) The design of a microprogram control unit with concurrent error detection - C. Y. Wong, W. K. Fuchs, J. A. Abraham, E. S. Davidson; FTCS-13*; pp. 476-483.

8.4) Am2910A microprogram controller; Bipolar Microprocessor Logic and Interface, 1985 Data Book; Advanced Micro Devices Inc.; Sunnyvale, California, USA; pp. 5/166-5/183.

8.5) Design of self-checking MOS-LSI circuits. Application to a four-bit microprocessor - Y. Crouzet, C. Landrault; FTCS-9*; pp. 189-192.

8.6) Design of self-checking MOS-LSI circuits. Application to a four-bit microprocessor - Y. Crouzet, C. Landrault; IEEE Trans. Comp.; Vol. C-29, No. 6; June, 1980; pp. 532-537.

8.7) A monolithic self-checking error detection processor - J. Chavade, M. Vergniault, P. Rousseau, Y. Crouzet, C. Landrault;1980 Test Conf.*;pp. 279-286.

8.8) Design specifications of a self-checking detection processor - Y. Crouzet, C. Landrault; FTCS-10*; pp. 275-277.

8.9) The PAD : A self-checking LSI circuit for fault-detection in microcomputers - J. Chavade, Y. Crouzet; FTCS-12*; pp. 55-62.

8.10) Fault-tolerance of a general purpose computer implemented by very large scale integration - R. M. Sedmak, H.L. Libergot; FTCS-8*; pp.137-143.

8.11) Implementation techniques for self-verification - R. M. Sedmak; 1980 Test Conf.*; pp. 267-278.

8.12) Design for self-verification: An approach for dealing with testability problems in VLSI-based designs - R. M. Sedmak; 1979 Test Conf.*; pp. 112-120.

8.13) Error detecting codes, self checking circuits and applications - J. F. Wakerly: Elsevier - North Holland; New York; 1978; Chapter 8, pp. 175-206.

8.14) A study of standard building blocks for the design of fault-tolerant distributed systems - D. A.

Rennels, A. Avizienis, M. Ercegovac; FTCS-8*; pp. 144-149.

8.15) A research oriented microcomputer with built in auto-diagnostics - J. Moreira de Souza, E. Peixoto Paz, C. Landrault; FTCS-6*; pp. 3-8.

8.16) Fault-tolerant digital clocking system - J. Moreira de Souza, E. Peixoto Paz; Electronics Letters; Vol. 11, No. 18; September 4, 1975; pp. 433-434.

8.17) Cost effectiveness of self checking computer design - W. C. Carter, G. R. Putzola, A. B. Wadia, W. G. Bouricius, D. C. Jessep, E. P. Hsieh, C. J. Tan; FTCS-7*; pp. 117-123.

* see section B.5.

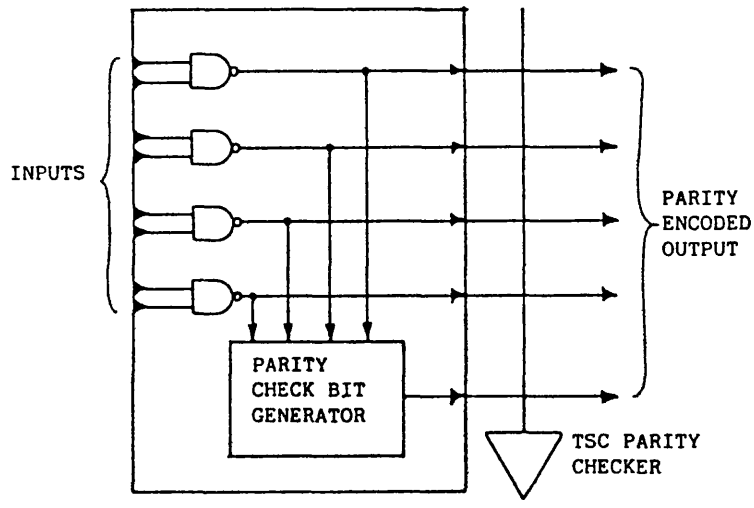


FIGURE 8.1 NAND PACKAGE WITH ENCODED OUTPUT

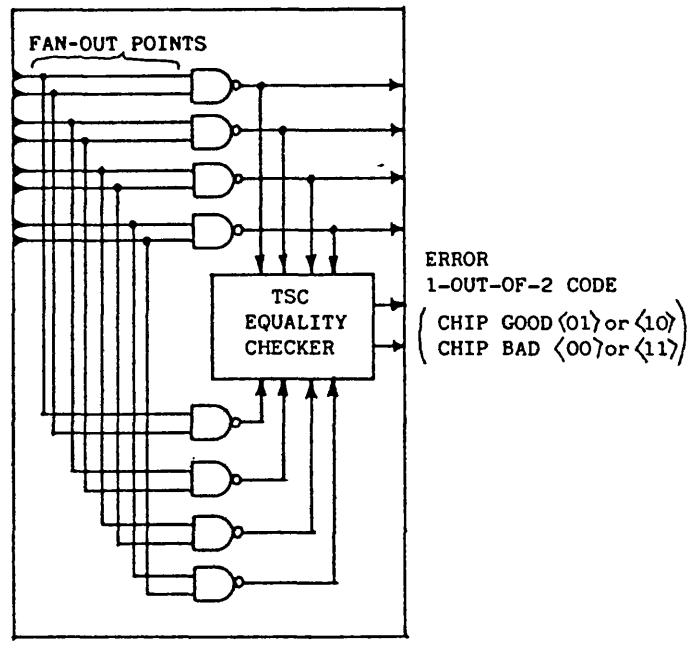


FIGURE 8.2 SELF CHECKING NAND PACKAGE

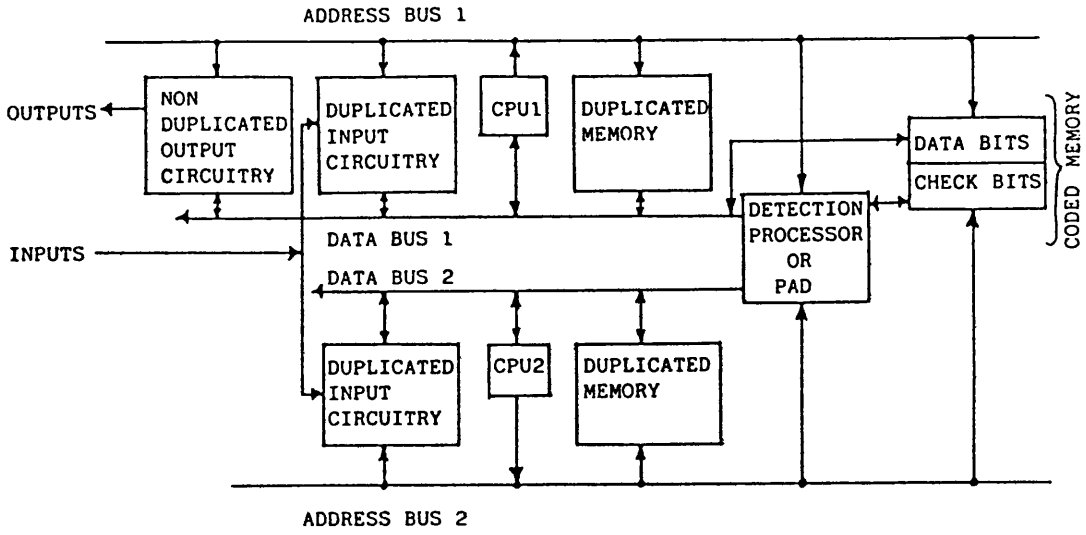


FIGURE 8.3 TYPICAL ARCHITECTURE OF A SYSTEM USING EITHER THE ERROR DETECTION PROCESSOR OR THE PAD

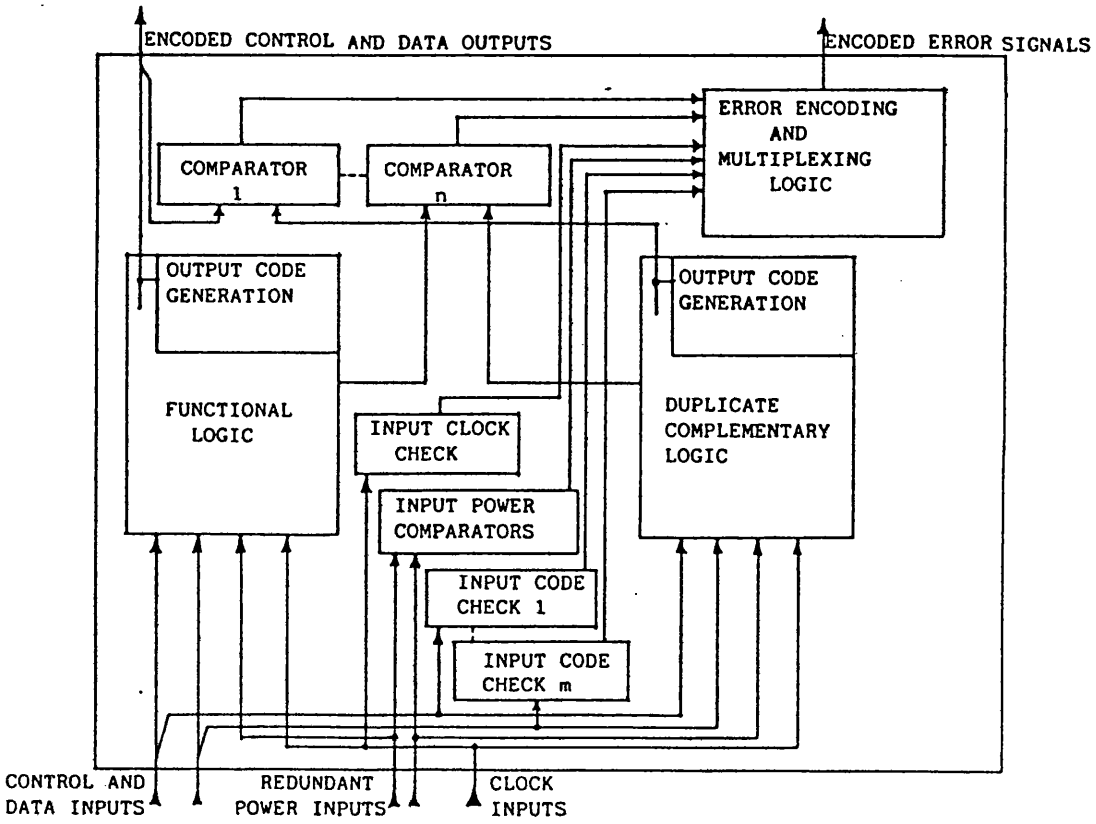
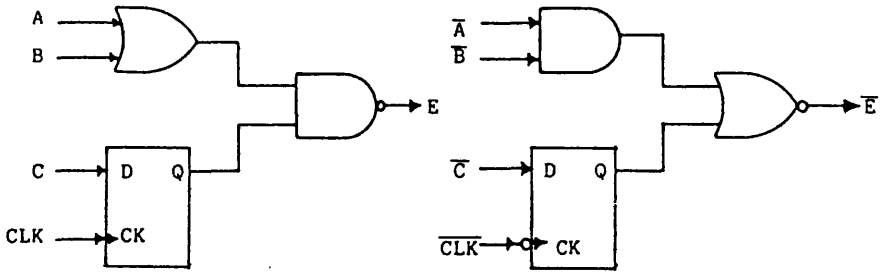


FIGURE 8.4 GENERALISED VLSI CHIP



(a) FUNCTIONAL CIRCUIT (b) COMPLEMENTARY CIRCUIT

FIGURE 8.5 COMPLEMENTARY LOGIC

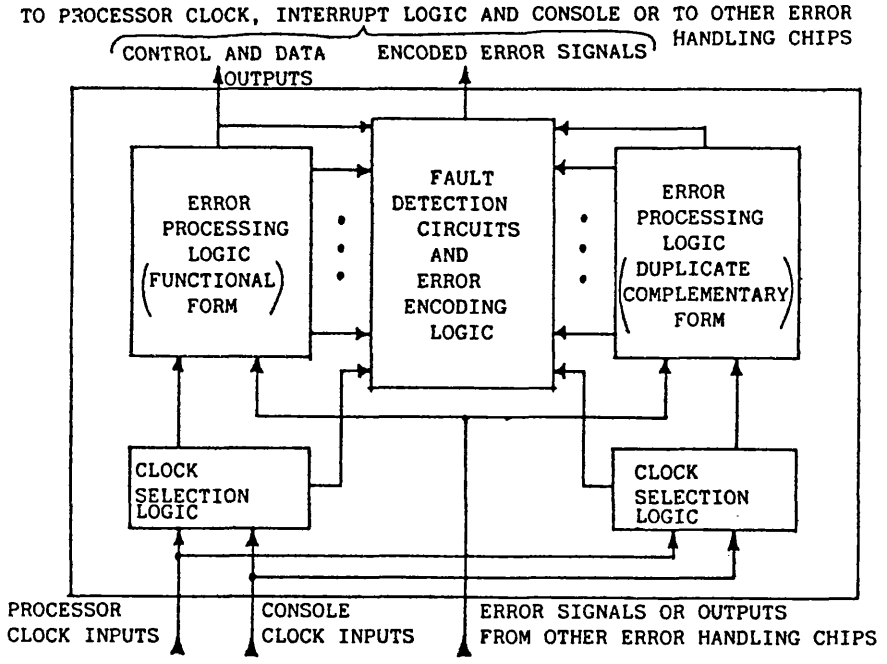


FIGURE 8.6 ERROR HANDLING CHIP

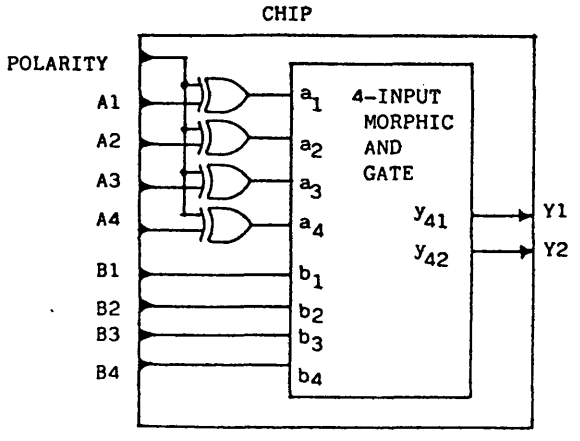


FIGURE 8.7 4-INPUT TSC COMPARATOR WITH INPUT POLARITY CONTROL

CHAPTER NINE : AN EXPERIMENTAL SELF CHECKING COMPUTER

9.1 : INTRODUCTION

An experimental self checking computer, described in this Chapter, brings together a substantial amount of the theory, ideas and designs presented in Chapters 5 to 8. A microprocessor based system is converted from being totally unchecked, to fully checked in certain specific areas. The two principal aims of this computer are to provide fault diagnosis to chip or circuit level, as appropriate, and to demonstrate the implementation of various self checking techniques.

A number of conditions are imposed on the unchecked and checked designs:

1) The unchecked microprocessor based system should be representative of typical designs and be implemented with widely used integrated circuits.
2) The use of complex circuitry should be avoided in the checked system.

The unchecked system is described in section 9.2. The checked system is not intended, at present, to be totally self checking, but section 9.3 discusses possible self checking techniques for every aspect of the unchecked system. Specific circuitry used in the checked system, is also detailed. The fault detection and error control circuits adopted in the experimental computer are described in section 9.4, whilst section 9.5 investigates several practical implementations of TSC comparators. Section 9.6 considers the testing of the resulting system.

9.2 : THE UNCHECKED SYSTEM

The experimental computer is a minimal microprocessor based system using components from the Motorola 6800 family [9.1]. It has three modules; central processing

unit (CPU), memory and input/output (I/O). Each module is housed on a separate board, see Fig. 9.1.

The following letters are appended to each signal name, principally to identify the bus signals within each board. The address bus, A0-A15, is given as an example in each case.

C - CPU board : eg. AC0-AC15
M - memory board : eg. AM0-AM15
I - I/O board : eg. AI0-AI15
B - backplane : eg. AB0-AB15

A description of each board in the unchecked system is given below.

The CPU board, illustrated in Fig. 9.2, consists of:

a) An MC6800 8-bit microprocessor [9.1].
b) An MC6875 2-phase clock generator [9.1], with crystal and reset switch.
c) One octal non inverting transceiver package, to buffer the data bus to or from the backplane. Its direction is controlled by the CPU board read/write line (R/$\overline{W}$C). It is enabled during the second clock phase (ϕ2 at 1).
d) Two octal non inverting line driver packages, to buffer the address bus to the backplane. They are enabled by the valid memory address (VMA) line.
e) One octal non inverting line driver package, to buffer the control bus to the backplane. It is permanently enabled.

The memory board, illustrated in Fig. 9.3, consists of:

a) One 2K (2048) words x 8-bit erasable programmable read only memory (EPROM). It is enabled by addresses in the range $E000\text{-}E7F7_{16}$ and $FFF8\text{-}FFFF_{16}$. It cannot be be enabled by a write operation. The EPROM contains the Motorola Minibug II monitor routines, with addi-

tional commands to load, print and move memory contents [9.2].

b) Two 1K words x 4-bit random access memories (RAMs). They are enabled by addresses in the range A000-$A3FF_{16}$. A memory write operation can only take place during $\phi 2$.

c) A 3-to-8 line decoder from address lines 13 to 15 (A13 -A15). The decoder provides enables for memory and I/O devices. It is enabled by VMA. Additional gates provide specific enables for the I/O devices.

d) One octal non inverting transceiver package, to buffer the data bus to or from the backplane. Its direction is controlled by R/$\overline{W}$M. It is enabled by addresses in the range A000-$BFFF_{16}$ and E000-$FFFF_{16}$, during $\phi 2$.

e) Two octal non inverting line receiver packages, to buffer the address bus from the backplane. They are permanently enabled.

f) One octal non inverting line receiver package, to buffer the control bus from the backplane. It is permanently enabled.

The I/O board, illustrated in Fig. 9.4, consists of:

a) An MC6850 Asynchronous Communications Interface Adaptor (ACIA) [9.1] with an RS232 compatible line driver and line receiver as a terminal interface. The ACIA is enabled by addresses 8008_{16} and 8009_{16}.

b) An MC14411 Bit Rate Generator [9.1] and crystal. This provides the ACIA transmit and receive clocks, such that the ACIA operates at 300 BAUD.

c) An MC6821 Peripheral Interface Adaptor (PIA) [9.1]. Peripheral data lines PB0-PB7 and peripheral control lines CA2 and CB2 are assigned as outputs. They each drive light emitting diodes (LEDs). Peripheral data lines PA0-PA7 are assigned as inputs. A set of switches provide a 0 or a 1 to each input. The PIA is enabled by addresses 8004-8007_{16}.

d) One octal non inverting transceiver package, to buffer the data bus to or from the backplane. Its direction

is controlled by R/$\overline{W}$I. It is enabled by addresses in the range 8000-9FFF$_{16}$, during ϕ2.

e) Two octal non inverting line receiver packages, to buffer the address bus from the backplane. They are permanently enabled.

f) One octal non inverting line receiver package, to buffer the control bus from the backplane. It is permanently enabled.

9.3 : THE CHECKED SYSTEM

This section discusses self checking techniques for all aspects of the unchecked system. In certain cases, the specific hardware used in the experimental computer is detailed.

9.3.1 : CPU

The CPU is duplicated and the outputs of the two processors compared, as shown in Fig. 9.5. This is the most convenient technique to adopt. A self checking processor, as proposed in sections 8.3.1 and 8.3.2, could be implemented using discrete gates and devices, but this is not practical or relevant to this experimental computer.

The compared outputs of the dual CPUs are the address and data buses, plus control lines VMA, BA and R/$\overline{W}$. The three comparators, shown in Fig. 9.5, are totally self checking (TSC). Their outputs can be merged into a single 1-out-of-2 error signal. The outputs of the functional CPU become the system signals. However, since the data bus is bidirectional, and therefore a set of inputs as well as outputs, information on this bus must be applied to both CPUs during a processor read operation (R/$\overline{W}$=1). This is achieved with the tri-state unidirectional buffer shown in Fig. 9.5. It is enabled during a processor read operation only.

No additional circuitry is required to synchronise the two

6800 CPUs. They have a common clock (ϕ1 and ϕ2 generated from the MC6875) and during normal operation perform identical operations from a reset condition.

The scheme of Fig. 9.5 could form the basis of a self checking microprocesor, as described in section 8.2 for a NAND gate package. All inputs would have to fan out to both processors from their respective chip pins, in order to minimise internal faults affecting both processors in the same manner. This has also been discussed in section 8.2. In addition, all inputs to both processors ideally need to be checked. The checking of clock, data bus, halt and reset inputs is considered in other sections. The rest of the inputs, $\overline{NMI}$,$\overline{IRQ}$, TSC and DBE, are not used and therefore connected to the appropriate logic level. These inputs are difficult to check, because they are static levels. They could individually, or as a group, be encoded with their encoding checked externally at the relevant chip pins. Alternatively, they could become part of the maintenance procedures for non self testing circuitry, as discussed in section 7.8.4. Mechanisms would be provided to exercise these normally static inputs and the resultant response of the processors checked. This process would be effected by a combination of hardware and software.

Power line failures also need to be considered, both within a chip and external to it. Short circuit decoupling capacitors, for example, cause a power line failure and are difficult to locate. Recently, however, fail safe capacitors have been introduced, which predominantly fail to an open circuit condition [9.3]. Power rails are often monitored for failures by light emitting diodes. The generalised VLSI chip described in section 8.3.7 has redundant power inputs, which are fed separately to the functional and duplicate logic. They are checked internally by comparison. However, in general, the technique of feeding redundant inputs separately to each of the two circuit copies and the internal checking of external

inputs, unless carefully implemented, should be avoided. This is because failures external to the chip will then cause it to generate an error signal. Replacing the chip under these circumstances will be futile, since there is nothing wrong with the chip itself. In the generalised VLSI chip, it is assumed that the occurrence of input and power failures can be specifically identified from the encoded error signals. Static inputs and power rails are not checked in the experimental computer.

9.3.2 : System and CPU Clocks

The MC6875 clock generator provides the CPU clocks, MPUØ1 and MPUØ2, plus the system clock, BUSØ2. The BUSØ2 clock signal is checked with the TSC periodic signal checker described in section 6.9.

Fig. 9.6 shows the output circuitry of the MC6875, from which it can be seen that MPUØ1 and MPUØ2 are the Q and $\bar{Q}$ outputs of a T-type flip-flop, whilst BUSØ2 is MPUØ1 inverted. BUSØ2 is fully checked for period, mark-space and stuck-at faults. If this signal is correct, then MPUØ1 cannot have period or mark-space faults. The only fault which can affect it is a wire break between the fan-out point of Q (see Fig. 9.6) and its integrated circuit pin. This will give the appearance of a stuck-at fault to the checker. Similarly, if BUSØ2 is correct, then it is unlikely that MPUØ2 will be a signal of different period or mark-space ratio, since it is generated from the same flip-flop. Therefore, only stuck at faults are considered possible on MPUØ2.

If BUSØ2 is fully checked, it is thus unnecessary to fully check MPUØ1 and MPUØ2. These two signals only need to be checked for stuck-at faults. Since MPUØ1 and MPUØ2 naturally form a 1-out-2 encoded pair, this is achieved by connecting them directly to the fault indication circuitry. Fig. 9.6 shows the two checking mechanisms employed. If a fault in the output flip-flop of the

MC6875 causes ⟨MPUϕ1,MPUϕ2⟩ to become stuck-at ⟨01⟩ or ⟨10⟩, this not a problem, because the fault will be detected by the TSC periodic signal checker on BUSϕ2.

9.3.3 : Reset Line

Part of the MC6875 clock chip is used to provide power-on and manual reset facilities, as shown in Fig. 9.7. The reset line, $\overline{\text{RSTC}}$ on the CPU board, is normally static at 1. However, during a manual reset it is at 0. A simple fault detection mechanism is therefore a light emitting diode (LED), driven form the reset line via a buffer. This is illustrated in Fig. 9.7. The power-on reset might not be observed, but a manual reset will cause the LED to be lit for as long as the reset switch is depressed. If the LED does not indicate during this operation, or is permanently lit, the source of the failure can be easily traced to the LED, its driver, the reset line, the MC6875 or the reset switch. This circuitry is therefore self testing, assuming that the reset switch is depressed at some point during normal operation.

9.3.4 : Data Transmission Paths

Single bit odd parity is used for all data transmission in the experimental computer. It is chosen for the following reasons:

1) Minimum generation circuitry - a standard MSI integrated circuit.
2) Minimum checking circuitry - totally self checking EXOR trees.
3) Minimum redundancy.
4) Adequate error detection capability.
5) Odd parity is maintained by high impendance buses.

The last two reasons are expanded in subsequent paragraphs.

All the line driver, line receiver and transceiver integrated circuit packages used in the experimental computer for data transmission are eight bits wide; i.e. they contain eight of each particular device. A parity bit is therefore added to each group of eight, or up to eight lines. The data bus, address bus and a number of control lines grouped together as a control bus, are all parity encoded as follows:

1) Data Bus : 1 parity bit, DP, covering data lines D0-D7.
2) Address Bus : 2 parity bits; AP1 covering address lines A0-A7 and AP2 covering address lines A8-A15.
3) Control bus : 1 parity bit, CP, covering control lines $\phi 2$, VMA, R/$\overline{W}$ and $\overline{RST}$.

The parity bit for each group of eight, or up to eight signals is processed by an independent buffer, as indicated in Fig. 9.8.

Single bit parity will detect all single bit faults in a word. It will also detect all faults affecting an odd number of bits. A non transforming circuit for parity encoded data, such as Fig. 9.8, is self testing for all faults affecting less than all bits. This means that an enable or power line failure, which causes an all 1 or an all 0 output from the parity bit buffer or the data bit buffers, will eventually be detected. If tri-state buffers are used in the circuit of Fig. 9.8, then tie-up resistors will maintain a correct odd parity word (nine 1's) on the bus when the buffers are in their high impedance state. A similar situation occurs if a failure in the direction line of a bidirectional buffer package causes two sets of buffers to drive a bus.

The parity generation and checking used for each board is now considered.

CPU Board : Four parity bit generators are required, as

indicated in Fig. 9.9. These are all 74LS280 9-bit odd/even parity generators/checkers [9.4]. Input data to each generator must be checked, because the generator is a non code disjoint circuit; it will produce a correct parity bit for erroneous input data. The address bus, data bus, R/$\overline{W}$C and VMAC are all checked by CPU comparison, whilst ϕ2C and $\overline{RST}$C are checked by the mechanisms described in sections 9.3.2 and 9.3.3 respectively.

The parity generator for the data bus also requires a tri-state buffer at its output, as shown in Fig. 9.9. This is enabled for a processor write operation only (R/$\overline{W}$C=0). During a processor read operation, data bus parity is generated, or supplied by the transmitting device.

All the parity generation circuits described above could be included in the scheme of Fig. 9.5 for a self checking microprocessor. The chip would then not only indicate an internal failure, but have parity encoded outputs as well.

The three buses are checked on the CPU board with TSC EXOR trees, as described in section 6.7.

I/O Board : Both the ACIA and PIA require parity generation for the data bus during a processor read operation. This is achieved with a 74LS280 parity generator and a tri-state buffer, as shown in Fig. 9.10. The buffer is enabled for a processor read operation from that particular I/O device only, since other devices will supply parity when they are read and the CPU board supplies parity during a processor write operation. Parity is checked on all three buses with TSC parity checkers.

Fig. 9.10 assumes that each I/O device is isolated from others on the I/O board, so that the scheme presented is repeated for every I/O device. This is true if the isolating circuits are fully implemented, as suggested in section 9.2.8. The I/O board then becomes two sub-modules, one for the ACIA and one for the PIA. An

alternative structure is shown in Fig. 9.11, where the ACIA and PIA share a single data bus parity generator. The tri-state buffer connected to the output of the parity generator, is now enabled when the processor reads either the ACIA or the PIA; i.e. an I/O board read.

Memory Board : The memory board data, address and control buses are all checked using TSC parity checkers. The use of parity within the memory itself, is considered in the next section.

Parity is also checked on all three buses of the back-plane.

9.3.5 : Memory

The EPROM used is 8-bits wide, whilst the RAM chips are 4-bits wide. A single parity bit appended to each data word is therefore inadequate to check many internal decoder (addressing) failures. Three possible solutions are:

1) Use an 8-bit byte error detecting code for the EPROM and a 4-bit byte error detecting code for the RAM.
2) Replace both types of memory with equivalent bit-sliced devices.
3) Store the address parity bits with each data word.

The first solution is not adopted because of the added complexity of circuitry required; a) for checking the memory codes and b) for code translation between bus and memory codes (see section 7.3.3) The additional redundant memory required for this solution is, however, not a problem. The second solution, whilst being suitable for microprogrammed memory, is not so for conventional micro-processor based systems. It is rare, for example, to find bit-sliced EPROMs employed in such a system.

The third solution is the one adopted in the experimental computer, principally because the required parity bits are

already available. However, limitations of this technique using parity have already been considered in section 7.3.2.

A second 2K words x 8-bit EPROM is added to the system to hold the necessary check bits. This is referred to as the check EPROM, whilst the existing EPROM is referred to as the data EPROM. The control lines and address lines to this check EPROM are identical to those used for the data EPROM. Data and address parity bits are generated from the data stored in the data EPROM and the address used to access this data. This information is then loaded into the check EPROM, before it is installed into the system. Additional hardware is required to process its parity bits and this is shown in Fig. 9.12.

The data line of the check EPROM used for data parity, DPME, is connected directly to the data parity line of the memory board, DPM. It is simply an additional line for the data bus and therefore treated as such. There is no parity generation required for either type of memory. During a processor read from EPROM, the address parity bits in the check EPROM, APME1 and APME2, need to be compared with corresponding address parity bits generated on the CPU board. These are APM1 and APM2 respectively on the memory board. This is achieved with the 2-bit TSC equality checker shown in Fig. 9.12. Two tri-state buffers are also required and these are enabled during a processor read from EPROM, allowing APM1 and APM2 to be presented to the equality checker. There they are compared with APME1 and APME2 from the check EPROM. At all other times, outputs APME1 and APME2, as well as outputs from the two tri-state buffers, are in a high impedance state. Tie-up resistors on all these four outputs maintain identical inputs (1's) to the comparator. If this mechanism fails, or a mismatch of address parity bits occurs when the EPROMs are enabled, then the TSC equality checker will indicate an error. Note that during normal operation, both EPROMS are not enabled by a processor

write cycle.

A third 1K words x 4-bit RAM is added to the system to hold the three parity bits. This is referred to as the check RAM, whilst the existing RAMs are referred to as the data RAMs. The control and address lines to the check RAM are identical to those used for the data RAMs. The additional hardware required to process the parity bits is shown in Fig. 9.13. As for the check EPROM, the check RAM data parity line, DPMR, is connected directly to the parity line of the memory board, DPM. Two tri-state buffers are connected between the address parity bits of the memory board, APM1 and APM2, and the corresponding check RAM address parity bits, APMR1 and AMPR2. A TSC equality checker compares the inputs and outputs of these two buffers, as shown in Fig. 9.13. During a proccessor read operation from RAM, the buffer outputs are disabled, so the equality checker compares APM1 and APM2 with APMR1 and APMR2 respectively. At all other times, including a processor write to RAM, the buffers are enabled, so the equality checker compares identical signals. If these buffers fail, or a mismatch of parity address bits occurs when the RAM contents are read, then the TSC equality checker indicates an error.

If the isolation circuits are fully implemented within the system, both RAM and EPROM will become sub-modules of the memory board, as indicated in the previous section for I/O devices.

9.3.6 : Address Decoding

The 3-to-8 line decoder used for address decoding produces a 1-out-of-n encoded output when enabled. However, as indicated in section 6.8, if the enable is also included as part of the output, then a 1-out-of-n code output is always produced. Note that the 1-out-of-n code is active low in this instance, i.e. a single 0 in n bits.

The effects of single failures within this type of decoder have already been considered in Fig. 6.36. On this basis, the 1-out-of-n code checker described in example 6.7 is adequate for checking the decoder outputs. The design of example 6.7 is expanded in Fig. 9.14 to check a 1-out-of-9 code. No modifications are required to check an active low 1-out-of-n code. The decoder and its checker could form the basis of a self checking decoder integrated circuit.

The additional gates used to generate the ACIA and PIA enables, $\overline{ENA}$ and $\overline{ENP}$ repectively, are best checked by duplication and comparison, as shown in Fig. 9.14. The duplicate circuit can be constructed in normal or complementary logic. Alternatively, these gates can be replaced with a 2-to-4 line decoder, which has AM2 and AM3 as inputs and $\overline{EN8/9}$ as an enable. This decoder can then be checked using the 1-out-of-5 checker, described in example 6.7. This scheme uses less packages than the original gates with duplication and comparison. Only the main decoder is checked in the final form of the experimental computer.

9.3.7 : I/O Devices

The ACIA again uses the duplication and comparison technique employed for the CPU. The circuit for the ACIA is given in Fig. 9.15. A tri-state unidirectional buffer is also required, enabled this time by a processor write operation to the ACIA ($R/\overline{WI}=0$ and ACIA enabled). Both devices then receive identical data information. Only the functional ACIA drives the data bus during a read operation. Outputs compared with TSC equality checkers are the data bus (D0-D7), plus the $\overline{RTS}$, $\overline{IRQ}$ and Tx Data signals.

The MC14411 bit rate generator has sixteen different output clock rates. One of them is connected to both the transmit and receive clock inputs of the ACIA (Tx Clk and Rx Clk respectively). This clock signal is checked by the

TSC periodic signal checker described in section 6.9, as shown in Fig. 9.16. The monostable periods of the checker are adjusted to match the selected clock signal.

Input and output circuitry for the ACIA, which in this case is an RS232 compatible receiver and driver respectively, can be duplicated and compared if this is desired, see Fig. 9.17a. An alternative arrangement is shown in Fig. 9.17b, which assumes an access to, at least, the transmit data output and receive data input of both ACIAs. This is possible in the experimental computer, but not if the ACIA is constructed as indicated in Fig. 9.15. For a given fault, Fig. 9.17a would also, in general, produce a a different set of fault indications to Fig. 9.17b.

9.3.8 : Control Logic

The self checking computer has various individual gates, or combination of gates, which process at least two control signals (this includes enables), to produce further control signals. In addition, a number of control bus signals are inverted on each board to match the required input levels of specific devices. Fig. 9.18 gives some examples of this control logic from the memory board.

All control logic must be checked in a self checking computer . In the experimental computer, all inputs to this logic are taken from the control bus and/or the address decoder outputs. Both the control bus and the address decoder outputs are checked (parity and 1-out-of-n respectively), so whilst the inputs to the control logic are not, in general, encoded, they are checked. If this circuitry uses gates from self checking logic packages, as proposed in section 8.2, then it is inherently checked. Alternatively, each circuit must be duplicated and compared, as suggested for part of the address decoding logic in section 9.3.6.

9.3.9 : Isolators

From section 7.8.2, isolation circuits should be installed as follows:

1) A unidirectional isolation circuit in series with every receiver, in a single transmitter to multiple receiver structure - Fig. 9.19a.
2) A unidirectional isolation circuit in series with every transmitter, in a multiple transmitter to single receiver structure - Fig. 9.19b.
3) A bidirectional isolation circuit in series with every slave transceiver 'A' bus connection, in a master transceiver to multiple slave transceiver structure - Fig. 9.19c.

In all cases, multiple must also include single; i.e. single transmitter to single receiver and single transceiver to single transceiver structures. In general, then, there must ideally be an isolating circuit between every transmitter (gate output) and receiver (gate input). On this basis, the experimental computer would require many unidirectional and bidirectional isolating circuits. This is clearly impractical, so a limited amount of these circuits are included for evaluation.

In the experimental computer, isolating circuits are placed between the backplane and the memory and I/O boards, as shown in Fig. 9.20. Unidirectional isolators are used for the control and address buses, with bi-directional isolators for the data bus. The circuits adopted are those shown in Figs. 7.25 & 7.26b and derived in Appendix B. Appendix B also presents a fault table for each isolator. The non inverting line receivers buffering the control and address buses from the backplane are replaced with inverting types, since the unidirectional isolators are inverting. The data bus requires an additional TSC parity checker. This is between its isolating circuits and the transceiver which buffers it to

or from the backplane, see Fig. 9.20. The bidirectional isolators have a common control line, the base drive for each transistor. Thus, in a similar manner to the transceivers, independent drive buffers are provided for the data bit isolators and the parity bit isolator, as shown in Fig 9.20.

The fault table for the bidirectional isolator in Appendix B demonstrates that both data paths must be used during normal operation to fully check each isolating circuit and its associated buffer. It also implies that there must be at least two data paths to achieve this. The data paths in this instance are the CPU board to the memory board and the CPU board to the I/O board.

9.4 : FAULT INDICATION AND ERROR CONTROL LOGIC

The requirements of the fault indication and error control logic, for diagnostic purposes, are as follows:

1) An indication of the output from every TSC checker at the falling edge of clock phase $\phi 2$. The fault detection circuits are therefore not monitored continuously. The falling edge of $\phi 2$ is chosen, because it terminates every processor cycle.
2) An ability to permanently indicate a fault until the fault indication logic is reset. Permanent faults will automatically do this, but a transient fault may occur during a single cycle only and never be observed.
3) An ability to halt the processor on detection of an error. This prevents an error propagating throughout the whole system, creating a misleading error indication.
4) A means of manually halting the processor. The fault indication logic can be observed without any system activity (see section 9.6).
5) A means of testing all non self testing parts.

These requirements are achieved with the circuitry described below.

The 1-out-of-2 encoded output pair from each TSC checker is processed in the manner of Fig. 9.21, where an EXNOR gate merges the output pair into a single line. An additional EXOR gate then provides a means of testing this and subsequent non self testing circuitry, as previously discussed in section 7.8.1. A switch and buffered control line are required to provide this test function. The output from this EXOR/EXNOR arrangement feeds a J-K flip-flop, which can be configured as a D-type flip-flop for cyclic fault indication, or configured as a clocked S-R latch for permanent fault indication. The configuration of the J-K flip-flop is controlled manually, by a switch. The flip-flop is clocked by $\phi 2$ and reset by $\overline{RST}$. Both of these system signals are already checked (see sections 9.3.2 and 9.3.3). A light emitting diode is used as the fault indicator, driven directly from the $\overline{Q}$ output of the flip-flop.

In addition to feeding its own error indication circuit, each checker output is also connected to an n-input Morphic AND gate, in the manner of Fig. 9.22. This converts all the 1-out-of-2 encoded pairs to a single 1-out-of-2 encoded pair. This final pair is then merged and latched, with the inclusion of a test gate, to produce a signal which can halt the processor, if this is desired. A switch shown in Fig. 9.22 controls this action. The processor may be halted at any time, via another switch. A light emitting diode indicates the state of the halt line. This combination of switch and LED also allows the halt line to be tested in a similar manner to the reset line. An additional LED could be connected to the the Bus Available (BA) line of the CPU, as this is directly affected by a halt operation. However, this is unnecessary, as a failure in either CPU will be detected by the comparison process.

9.5 : PRACTICAL IMPLEMENTATION OF TOTALLY SELF CHECKING COMPARATORS

Section 6.6 has given three structures for an n-input Morphic AND gate. These are a 2-level AND/OR structure, cascaded 2-input Morphic AND gates and cascaded 2-input Morphic AND gates with level merging. The following types of integrated circuit (IC) are now considered to implement these structures.

1) Single gates.
2) 74LS51 AND-OR-INVERT gates [9.4].
3) 4-input Morphic AND gate (see section 8.3.8).

Fig. 9.23 compares the number of IC packages required for an 8-input Morphic AND gate, constructed with the appropriate type(s) of IC for each structure.

The 4-input Morphic AND gate would require the minimum number of packages. Using the 74LS51 is the second best solution, in terms of the number of packages, but results in six logic levels. The cascaded 2-input Morphic AND gates with and without level merging both require the same amount of packages, in this example, but the former approach is achieved using less logic levels. The 2 level AND/OR solution is totally impractical. Aside from the 128-input OR gates, it requires 256 packages. An advantage of using SSI packages in TSC structures is that, in general, the design can be well checked for single package failures. Failures can occur in the 74LS51 and custom chips, which cause a pair of lines to become stuck at ⟨01⟩ or ⟨10⟩.

9.6 : TESTING THE EXPERIMENTAL COMPUTER

Possible checking mechanisms for each part of the experimental computer have been discussed in previous sections. Not all these proposals have been adopted in the practical system. Some sections have already detailed the actual

hardware added to the unchecked system. Fig. 9.24 shows the complete schematic for the self checking computer as implemented. Self checking techniques have been applied to specific areas of the system for evaluation, so the computer is not totally self checking. Exhaustively testing the complete system is therefore inappropriate. The areas which need to be tested are as follows:

1) System clock generation.
2) Reset (switch, logic and line).
3) Halt (switch and line).
4) Main address decoder.
5) Memory parity (EPROM and RAM).
6) Isolating circuits between backplane and memory and I/O boards.

Appendix B has demonstrated that fault diagnosis is assisted by fault indications when the system is inactive. The situation is effected by halting both CPUs. Halting a 6800 CPU causes its address bus, data bus and R/$\overline{W}$ line to be put into a high impedance state, whilst VMA is forced low. The system clock (ϕ2) remains active, as does the system reset line. Fig. 9.25 details the effects of halting the processor on the rest of the system. There are no active memory or I/O enables. Tie-up resistors force floating lines to a logic 1 state. As a result odd parity is maintained on the address and data buses. The system also enters a read mode (R/$\overline{W}$C=1). Parity continues to be generated throughout the whole system for the control bus, and on the CPU board for the address bus.

Fig. 9.26 details the faults detected by the CPU, reset line, halt line, system clock and main decoder checking mechanisms. It also gives the checker which will detect the faults in each case. Various faults were applied to these areas to confirm the contents of Fig. 9.26. In general, these were stuck-at faults, created by replacing a line transmitter with a 0 or 1 source. In all cases, the error control circuitry is set to halt the processor

on detection of an error. Timing faults were created in the clock generation circuitry by adjusting the oscillator frequency and also replacing the 6875 clock generator with an external clock source. The software used for test purposes is either the normal command routines of Minibug II [9.2], or specially written routines. These programs enable specific devices to be activated when required, or specific data to be transmitted on the three buses.

Fig. 9.27 details the faults detected in the memory circuits and by which checkers. Again, various stuck-at faults and words with incorrect parity confirm these results. A number of words with incorrect parity are stored in the EPROMs when they are programmed, for test purposes. RAM words with incorrect parity are obtained by reading memory which has not previously been written to, or storing words with incorrect parity.

Tests on the isolating circuits and their associated buffers are based on the fault analysis tables given in Appendix B. These fault tables are modified for use with the self checking computer in Figs. 9.28 and 9.29. All the single faults in these tables were simulated in the isolating circuits as they were developed, but are repeated in the overall system to confirm the checker indications. Both fault tables detail the faults detected when the CPUs are halted, which, again, are confirmed by practical fault simulation. However, since a number of signals in the control bus are not permanently at 1 when the CPUs are halted, faults detected during this condition are detailed separately for this bus in Fig. 9.29.

Overall, the level of fault diagnosis in the experimental computer is dependent on the level of fault detection, which in turn is dependent on the type of checking mechanisms used and the number of checkers provided.

The duplication and comparison of the CPU will detect most faults (all except those which produce an identical change

in the outputs of both CPUs) However, the fault cannnot be diagnosed to the functional CPU or its duplicate, so both must be replaced if an error is signalled from the CPU comparator (C5). Both devices would automatically be replaced if they were within a single self checking chip.

In the three categories of faults for the clock generation circuitry (period, mark-space and stuck-at faults) all faults are detected with the two checkers employed (C1 and C2). If only a subset of all faults is considered, then the checking mechanisms can be greatly simplified from those necessary to detect all faults. The predicted effects on the address decoder outputs due to all single internal faults, for example, has significantly reduced the complexity of the required 1-out-of-n checker, since it is only necessary for the checker to detect two output errors (two or zero active outputs).

The single bit parity code, used for the encoding of all buses in the experimental computer, will detect all faults which cause an odd number of erroneous bits in an encoded word. Assuming that all combinations of bits occur in both the data information bits and the parity bit at some time during normal operation, then all faults except the bus stuck at a codeword will be detected eventually. If all faults are to be detected immediately, then the encoding must be capable of detecting all bit errors.

The comments made above for a parity encoded bus also apply to the storage of parity encoded data in RAM or EPROM. The comparison of address parity bits, stored along side the data in memory, with generated address parity bits during a memory read operation allows all faults in the storage of these bits to be detected, as well as faults in the comparison process. The comparison process will also detect a number of internal memory addressing errors. If the addressing errors detected are inadequate, then a more sophisticated code than parity will be required for the stored address check bits.

The diagnosis of an isolator fault from the checker outputs to the buffer associated with that isolator is dependent on two important factors. Firstly, the observation of fault indications from a halted system, which assist the fault diagnosis, and secondly, the use of both data paths during normal operation (CPU board $\longleftrightarrow$ memory board and CPU board $\longleftrightarrow$ I/O board), to ensure that certain failures are detected. The latter factor requires that the system is operated in a particular manner for the detection of failures and is therefore an 'operation for test'.

Another operation for test used in the experimental computer, more as a confidence test rather than a necessary operation for the detection of failures, is the accessing of data from memory with known parity errors. The manual operation of the reset, halt and test switches are also operations for test.

9.7 : REFERENCES

9.1) M6800 microcomputer system design data, plus data sheets for MC6821 (DS 9435), MC6875 (DS 9485) and MC14411 (ADI-306); Motorola Semiconductor Products Inc.

9.2) Fault diagnostics for microprocessor based systems - T. J. Hollis; Project report for the degree of B.Sc. in Electrical Engineering; University of Bath, England; 1980; Chapter 4 and Appendix 8.

9.3) Fail-safe capacitors - G. Lloyd; Electronic Engineering; June, 1984; pp. 71-2.

9.4) The TTL book for design engineers - The engineering staff of Texas Instruments components group; Texas Instruments; Fourth European edition; 1980.

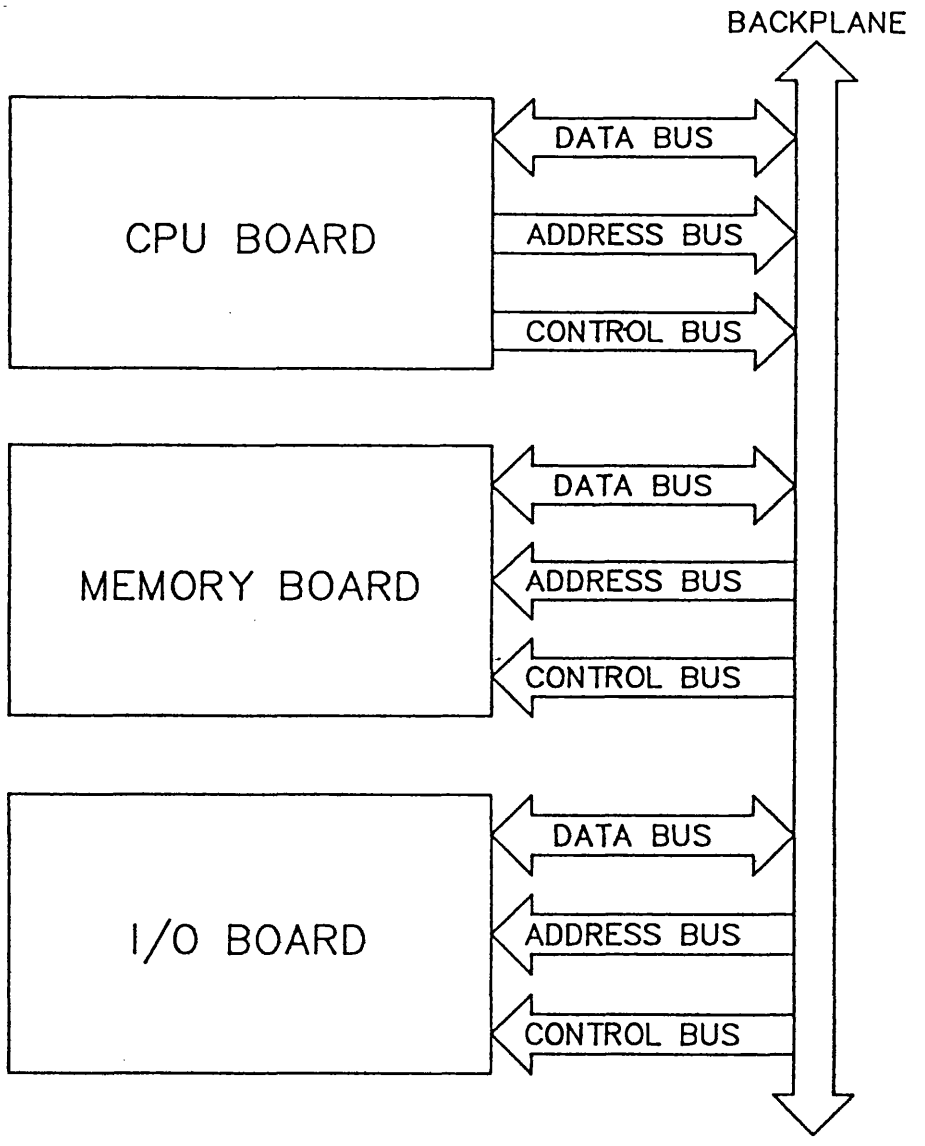


FIGURE 9.1 **BOARDS IN THE EXPERIMENTAL COMPUTER**

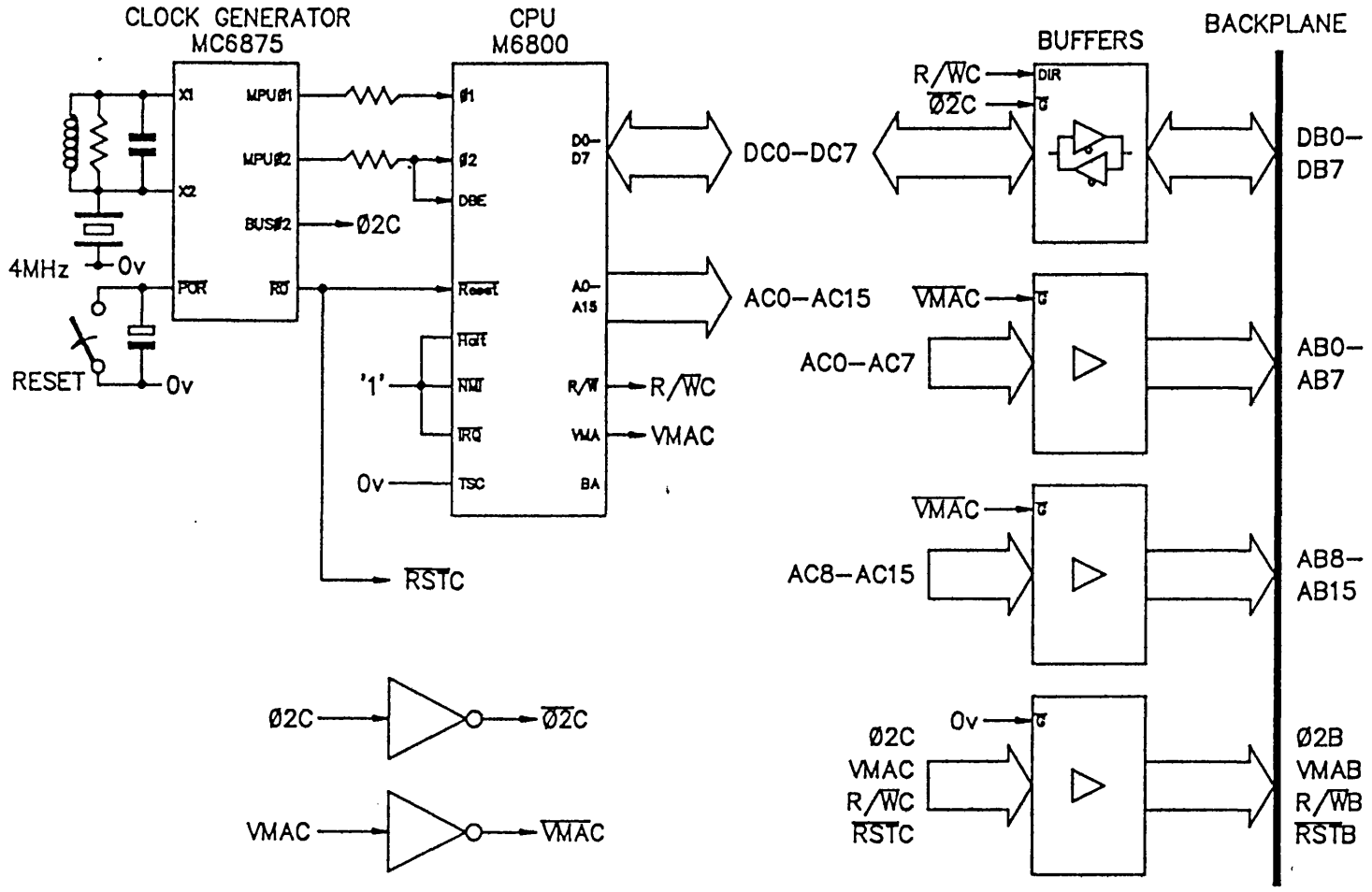


FIGURE 9.2 SCHEMATIC FOR THE UNCHECKED CPU BOARD

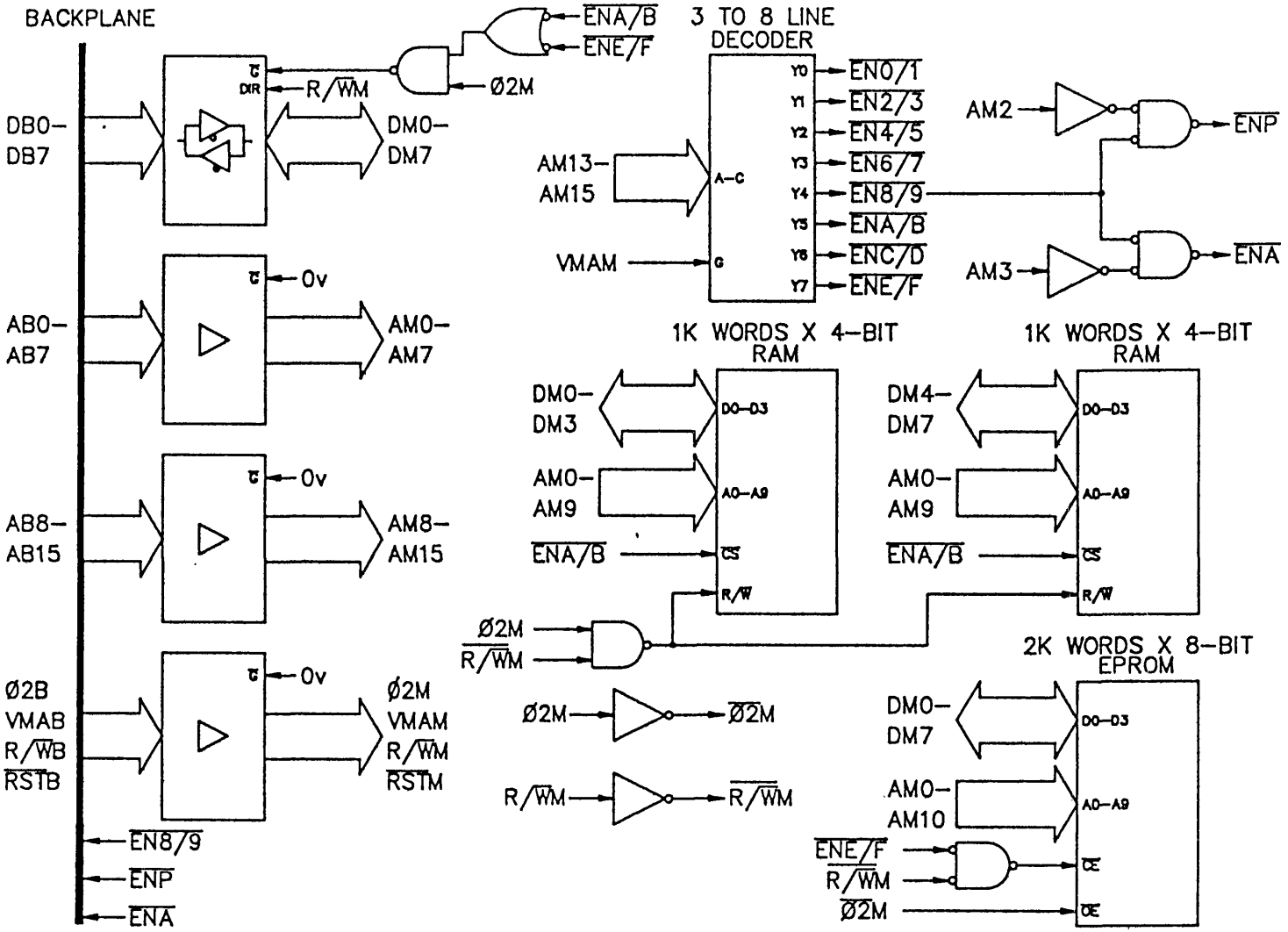


FIGURE 9.3 SCHEMATIC FOR THE UNCHECKED MEMORY BOARD

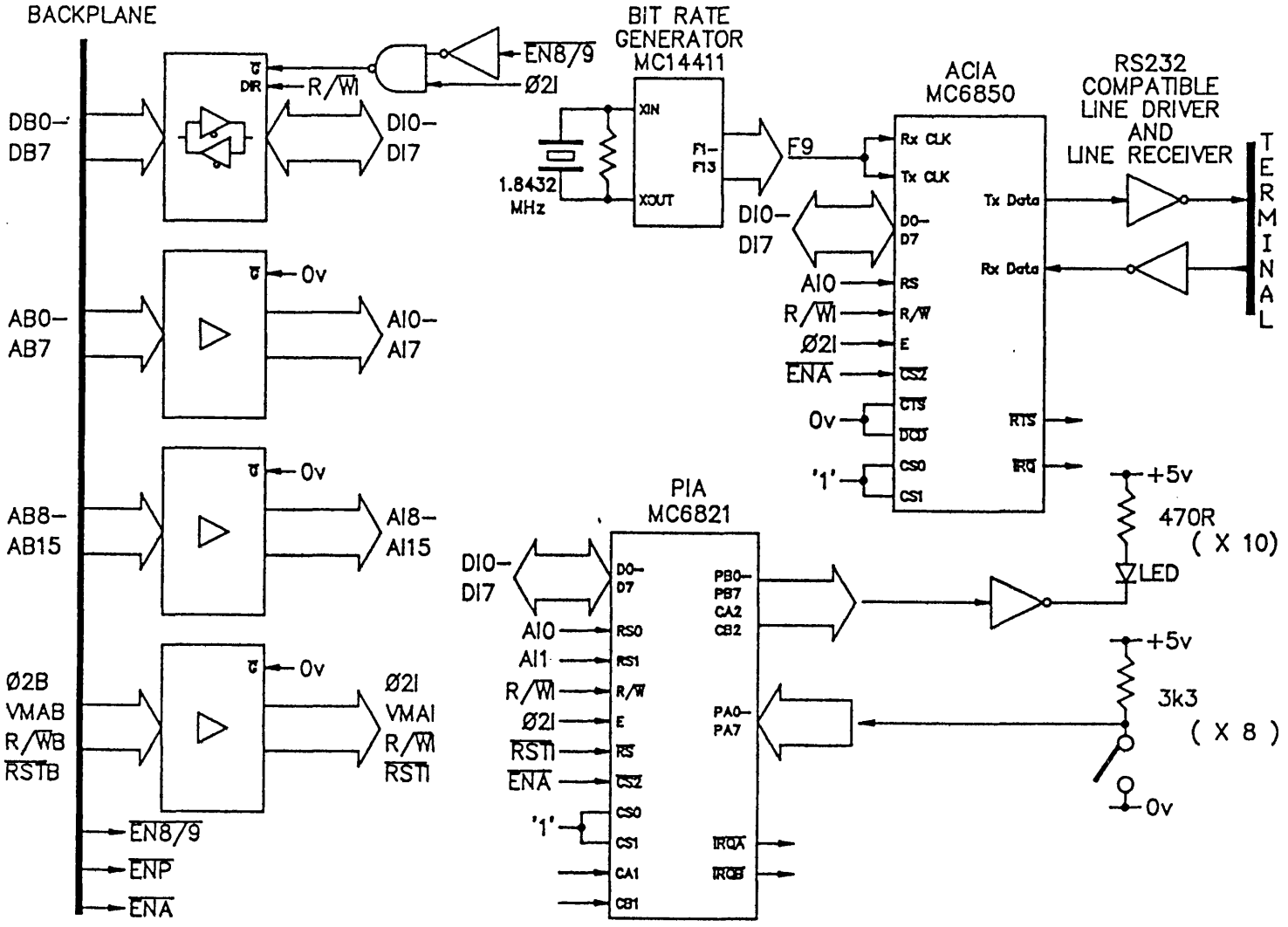


FIGURE 9.4 SCHEMATIC FOR THE UNCHECKED I/O BOARD

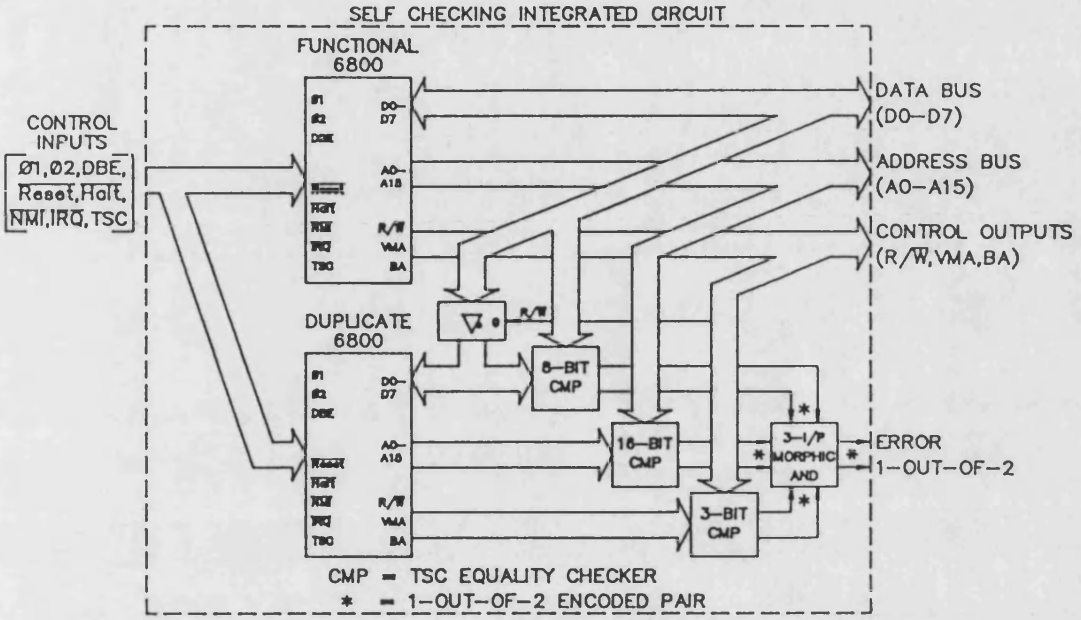


FIGURE 9.5 A SELF CHECKING 6800 MICROPROCESSOR

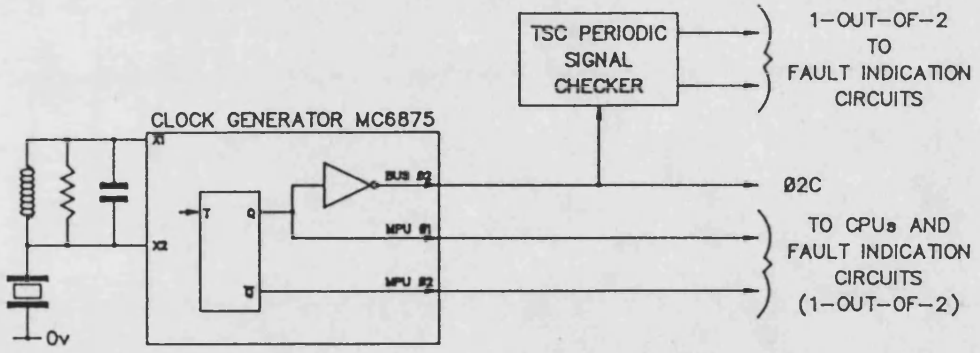


FIGURE 9.6 CHECKING THE SYSTEM AND CPU CLOCKS

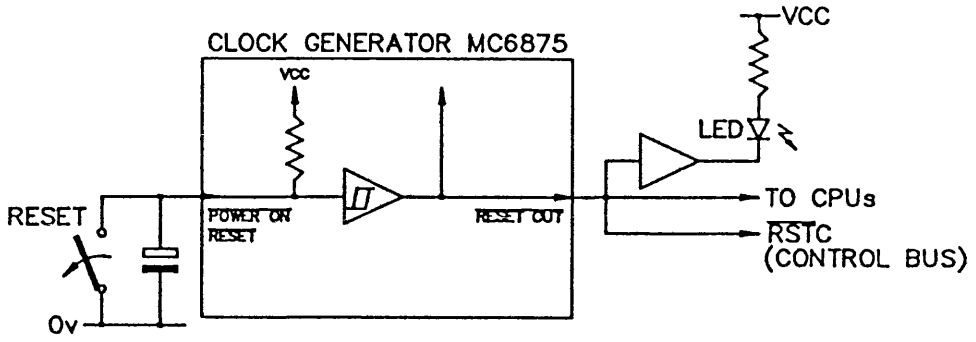


FIGURE 9.7 CHECKING THE RESET LINE

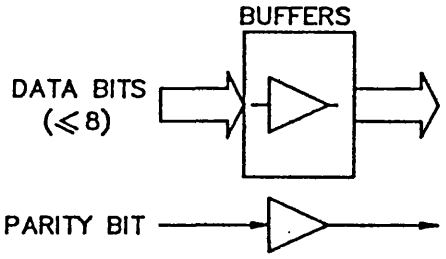


FIGURE 9.8 PROCESSING PARITY ENCODED DATA

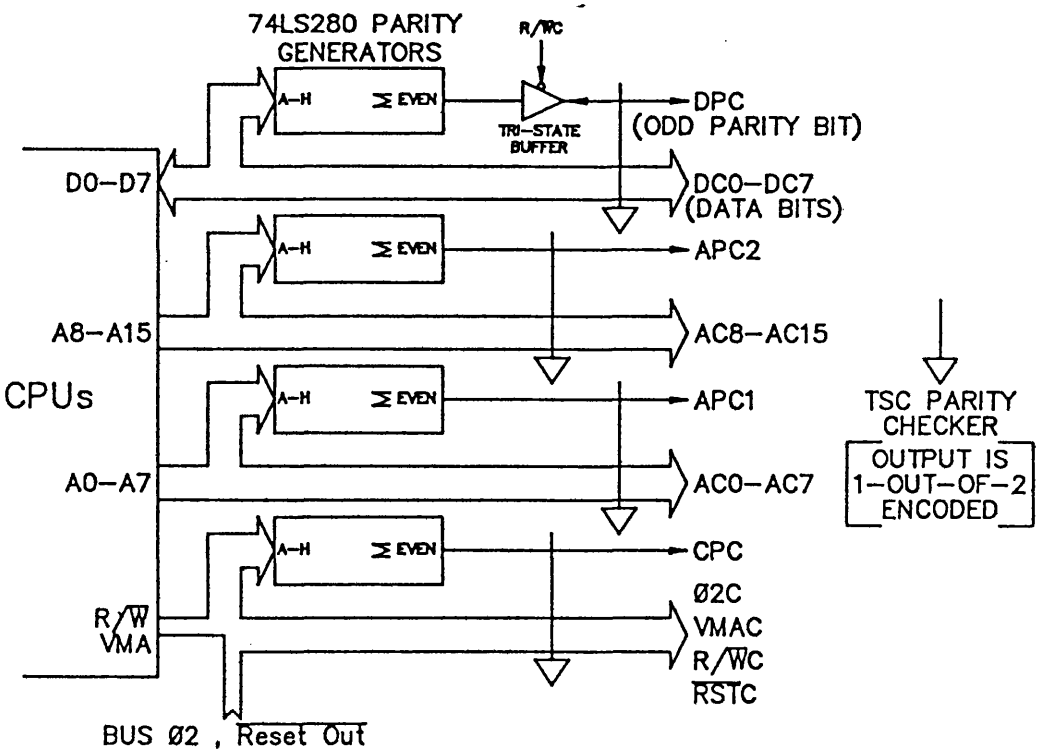


FIGURE 9.9 CPU BOARD PARITY

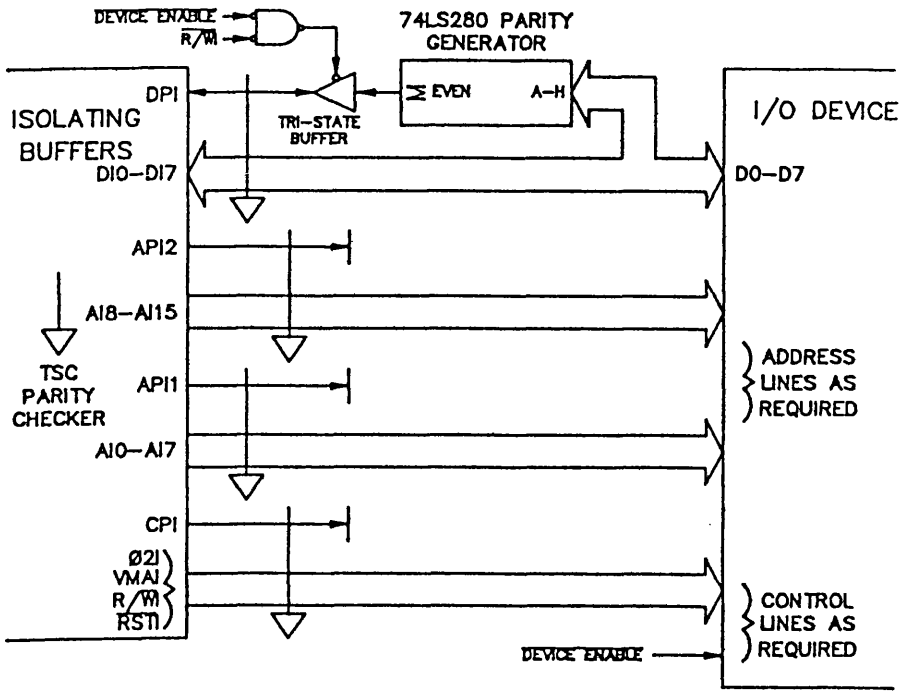


FIGURE 9.10 I/O DEVICE PARITY

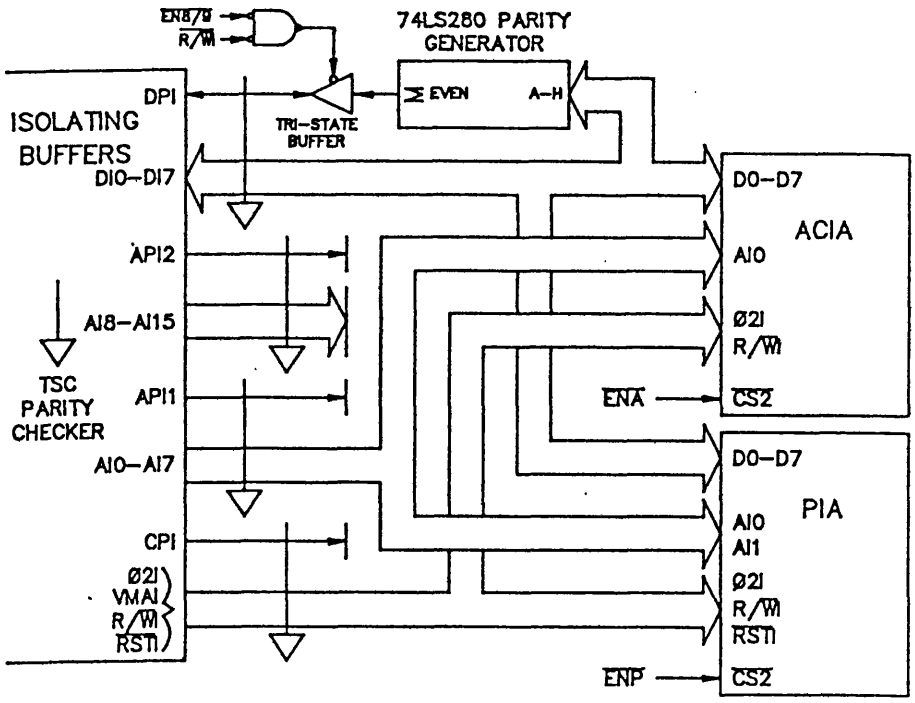


FIGURE 9.11 I/O BOARD PARITY

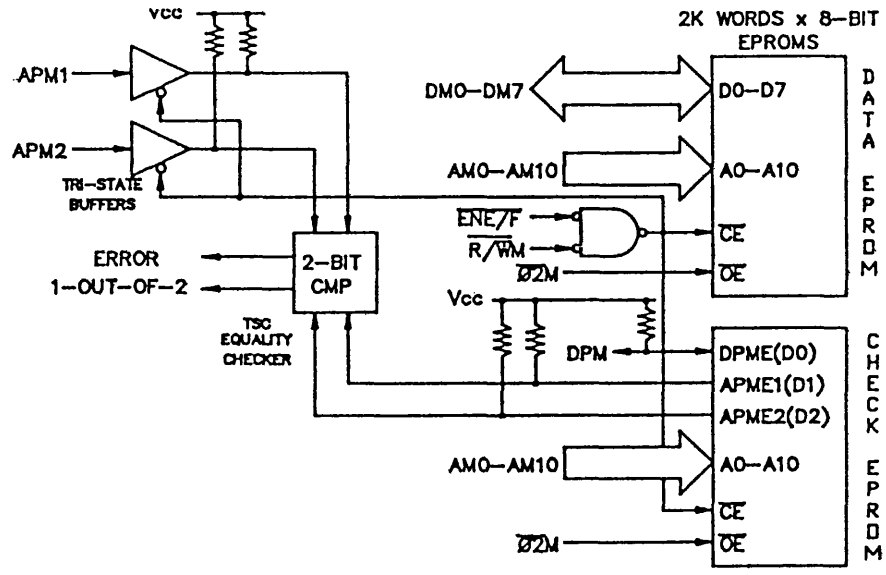


FIGURE 9.12 EPROM PARITY

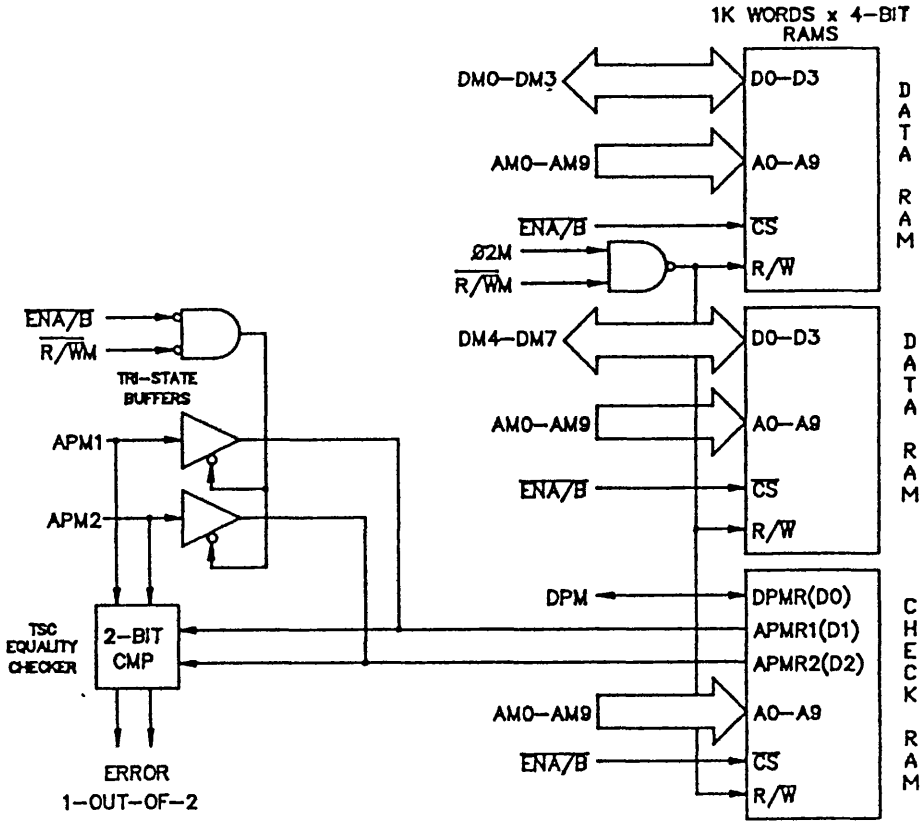


FIGURE 9.13 RAM PARITY

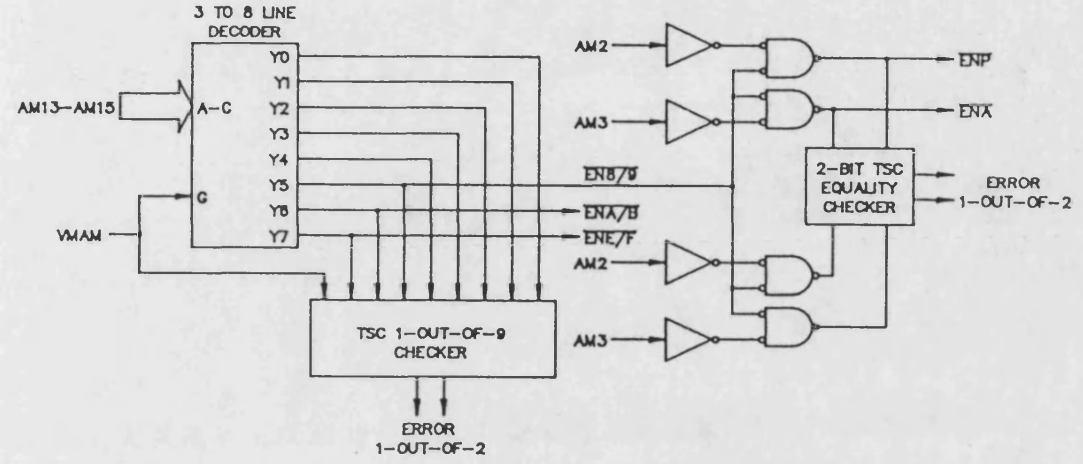


FIGURE 9.14 CHECKING THE ADDRESS DECODING

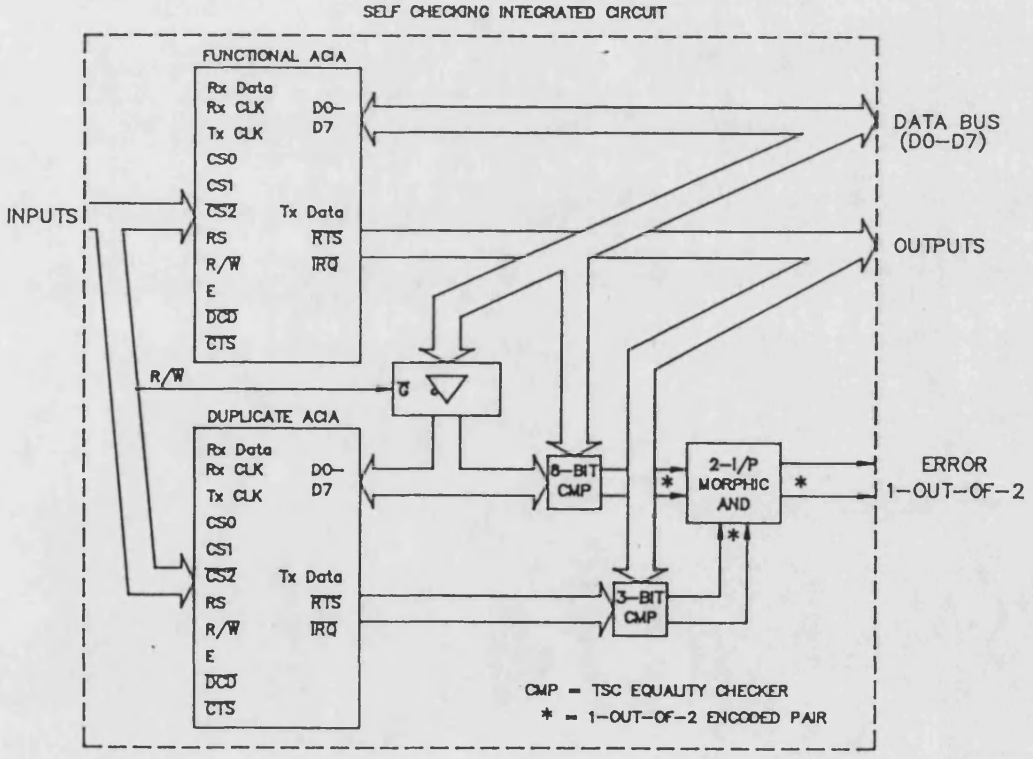


FIGURE 9.15 A SELF CHECKING MC6850 ACIA

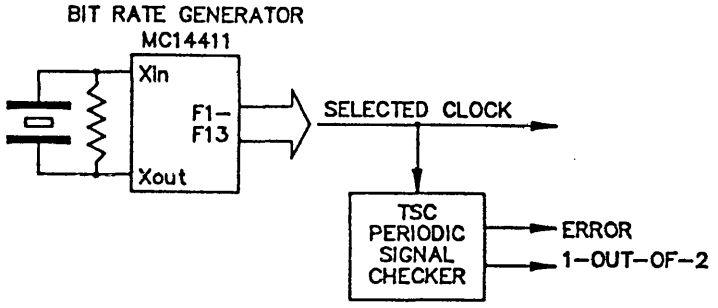
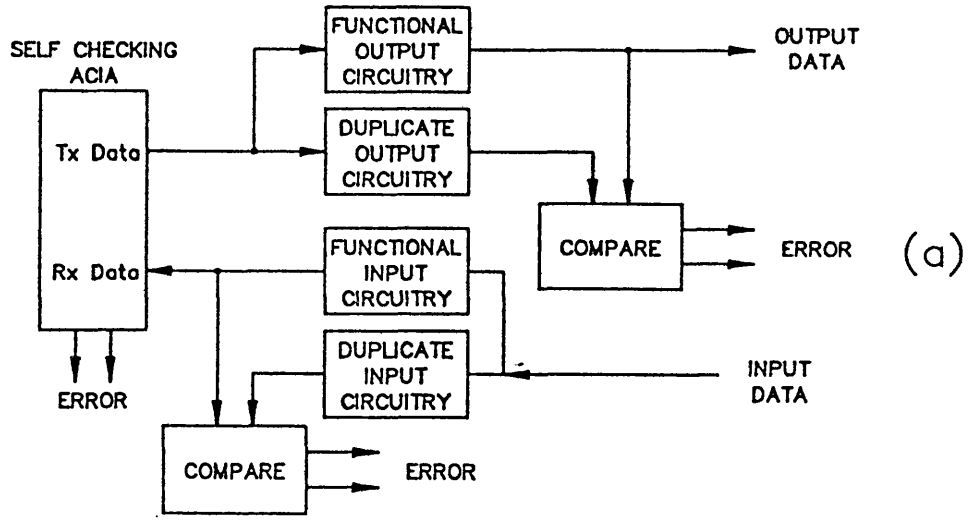


FIGURE 9.16 CHECKING THE ACIA CLOCK



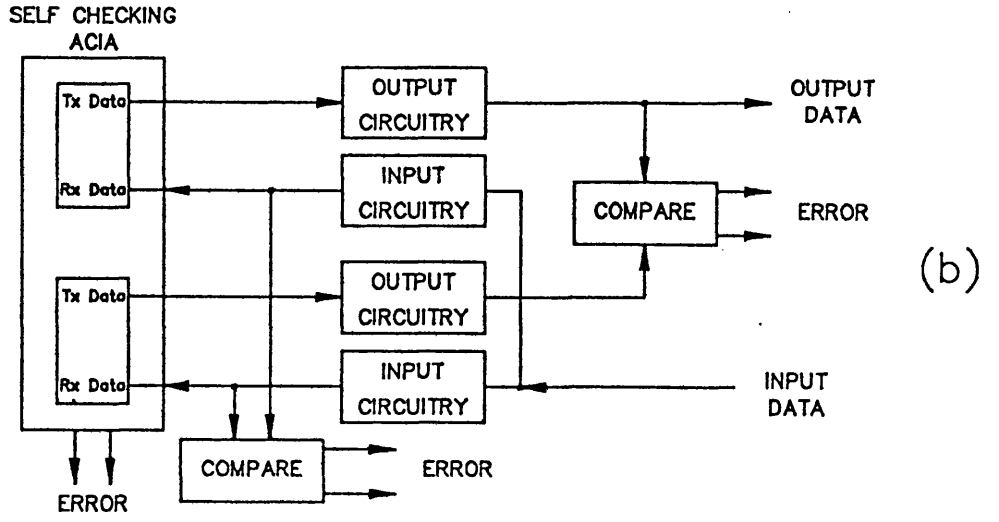


FIGURE 9.17 CHECKING I/O CIRCUITRY

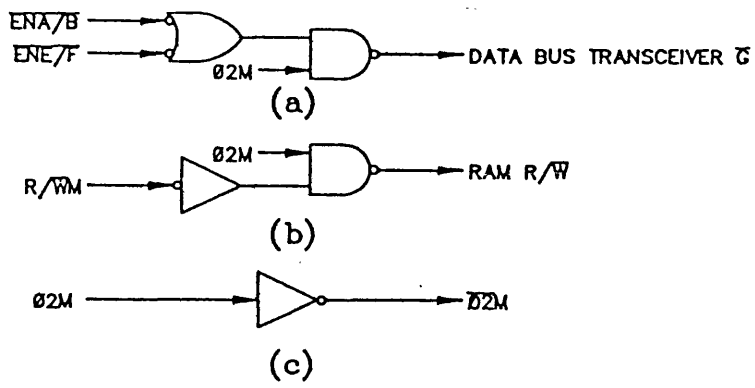
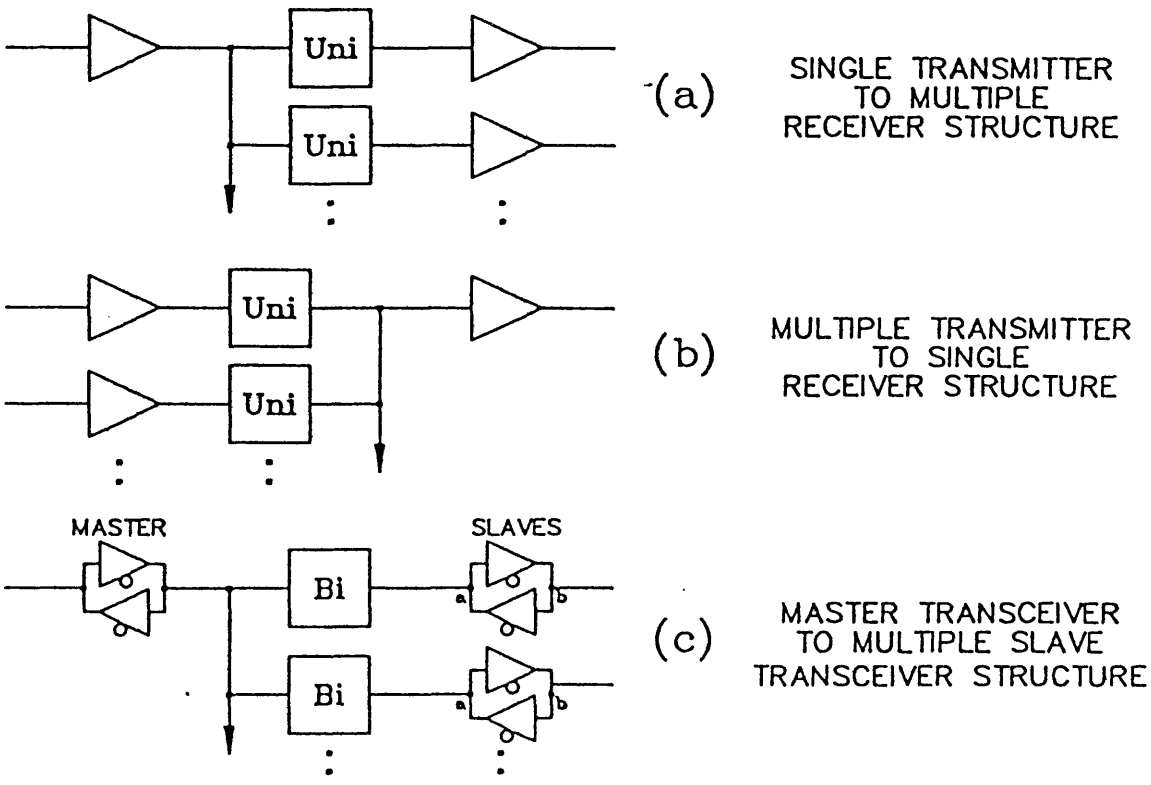
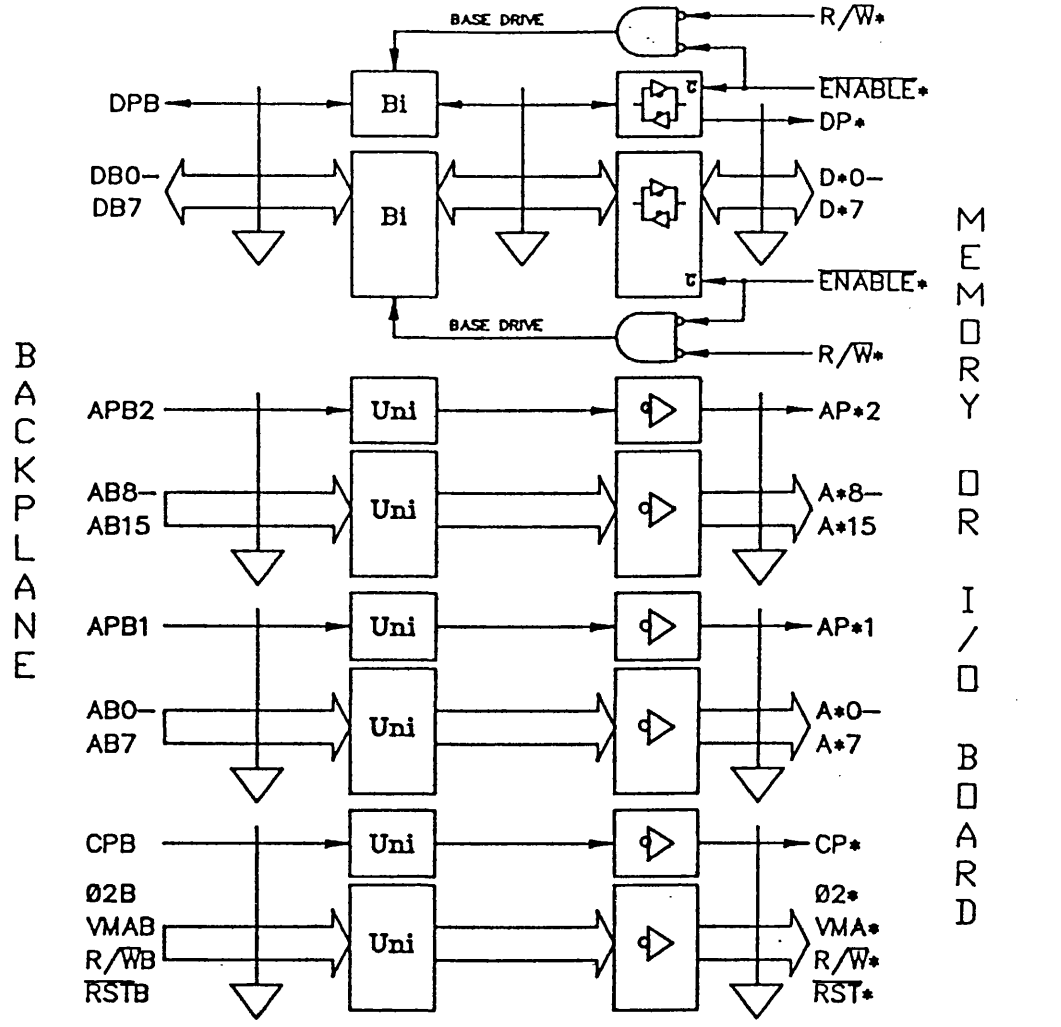


FIGURE 9.18 EXAMPLES OF MEMORY BOARD CONTROL LOGIC



Uni = Unidirectional Isolator Bi = Bidirectional Isolator

FIGURE 9.19 POSITIONING OF ISOLATORS



TSC PARITY CHECKER	Uni = Bidirectional Isolator Bi = Unidirectional Isolator	* = M (MEMORY BOARD) OR I (I/O BOARD)

FIGURE 9.20 ISOLATORS IN THE EXPERIMENTAL COMPUTER

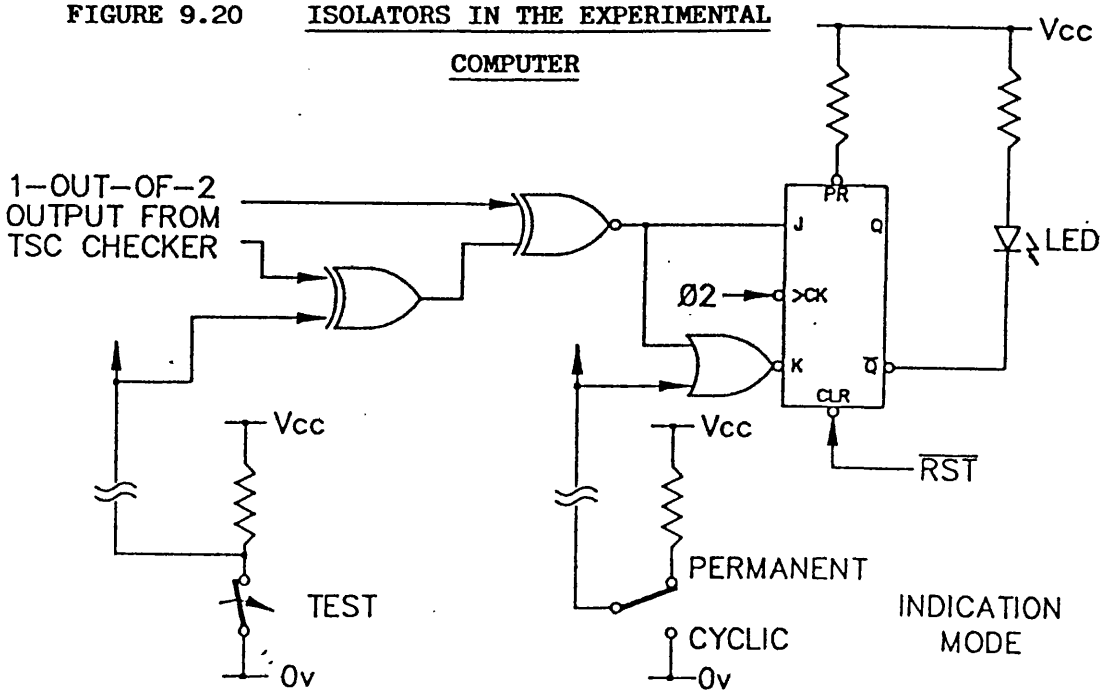


FIGURE 9.21 FAULT INDICATION LOGIC

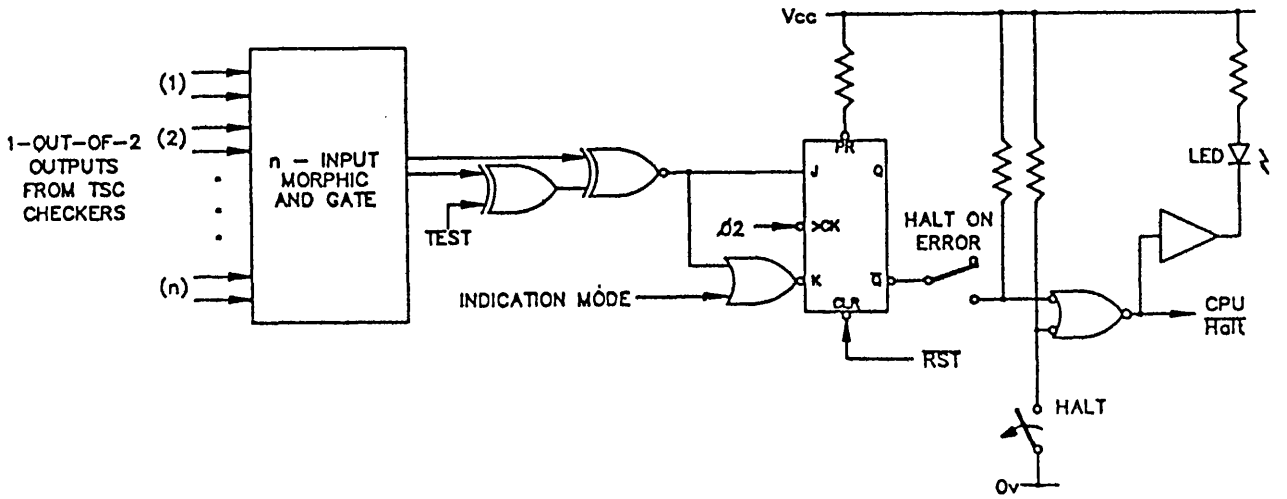


FIGURE 9.22 ERROR CONTROL LOGIC

STRUCTURE	INTEGRATED CIRCUIT TYPE		
	SINGLE GATES	74LS151	4-I/P MORPHIC AND
2-LEVEL AND/OR	3 LEVELS 256 x 8-I/P AND GATES 2 x 128-I/P OR GATES IMPRACTICAL IN THIS FORM	XX XX	CASCADED 4-I/P GATES 4-I/P GATE USES AND/OR STRUCTURE
CASCADED 2-I/P MORPHIC AND GATES (FIG. 6.27a)	6 LEVELS 24 x 2-I/P AND GATES 18 x 2-I/P OR GATES 10.5 PACKAGES	6 LEVELS 7 PACKAGES	4 LEVELS 3 PACKAGES
CASCADED 2-I/P MORPHIC AND GATES WITH LEVEL MERGING (FIG. 6.27b)	4 LEVELS 16 X 2-I/P AND GATES 2 X 2-I/P OR GATES 8 X 4-I/P OR GATES 4 X 4-I/P AND GATES 10.5 PACKAGES	XX XX	XX XX

FIGURE 9.23 INTEGRATED CIRCUIT PACKAGES FOR AN 8-INPUT MORPHIC AND GATE

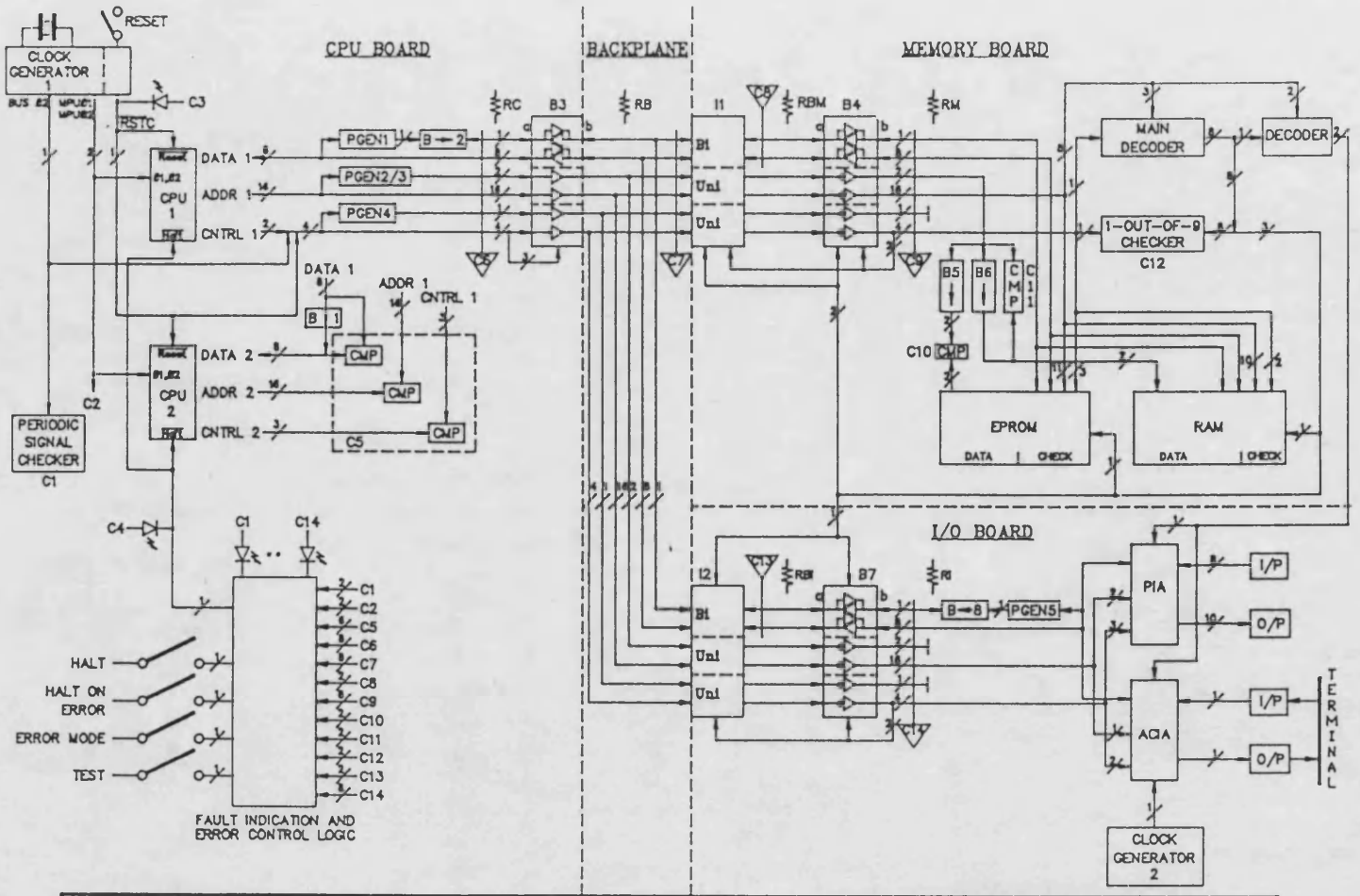


FIGURE 9.24 SCHEMATIC FOR THE EXPERIMENTAL SELF CHECKING COMPUTER

CPU BOARD		BACKPLANE	
DC0-DC7, DPC (φ2=0)	→ 1		
DC0-DC7, DPC (φ2=1)	= 1	DB0-DB7, DPB	→ 1
AC0-AC15	→ 1	AB0-AB15, APB1, APB2	→ 1
APC1, APC2	= 1	φ2B	⎍
φ2C	⎍	VMAB	= 0
VMAC	= 0	R/$\overline{W}$B	= 1
R/$\overline{W}$C	→ 1	$\overline{RST}$B	= 1
$\overline{RST}$C	= 1	CPB	⎍
CPC	⎍		

MEMORY BOARD		I/O BOARD	
DM0-DM7, DPM	→ 1		
AM0-AM15, APM1, APM2	= 1	DI0-DI7, DPI	→ 1
φ2M	⎍	AI0-AI15, API1, API2	= 1
VMAM	= 0	φ2I	⎍
R/$\overline{W}$M	= 1	VMAI	= 0
$\overline{RST}$M	= 1	R/$\overline{W}$I	= 1
CPM	⎍	$\overline{RST}$I	= 1
$\overline{EN0/1}$-$\overline{ENE/F}$, $\overline{ENA}$, $\overline{ENP}$	= 1	CPI	⎍

⎍ : CHANGING LEVEL
= 0 : DRIVEN LEVEL
= 1 : DRIVEN LEVEL
→ 1 : TIE-UP RESISTOR LEVEL

FIGURE 9.25 THE SYSTEM WITH CPUs HALTED

UNIT	DETECTABLE FAULTS	CHECKER DETECTING FAULTS
CPU	ALL FAULTS EXCEPT IDENTICAL FAILURES IN BOTH PROCESSORS	C5
RESET LINE	ALL FAULTS	C3
HALT LINE	ALL FAULTS	C4
CLOCK : BUS φ2	PERIOD, MARK-SPACE, STUCK-AT	C1
: MPUφ1,MPUφ2	STUCK-AT	C2
DECODER	NO ACTIVE OUTPUT TWO ACTIVE OUTPUTS (EVEN NO. OF ACTIVE OUTPUTS)	C12

(Checkers are those in Fig. 9.24)

FIGURE 9.26 VARIOUS SYSTEM FAULTS

DEVICE	DETECTABLE FAULTS	ERROR CREATED	CHECKERS DETECTING FAULT	CPU STATE
EPROM	ALL SINGLE AND MANY MULTIPLE FAULTS IN DATA AND DATA PARITY BITS	DATA PARITY	C9D etc.	READ
	ALL FAULTS IN ADDRESS PARITY BITS	ADDRESS PARITY	C10	READ
	ANY FAULT CAUSING THE EFFECTIVE INTERNAL ADDRESS TO BE AN ODD HAMMING DISTANCE FROM THE DESIRED ADDRESS IN ITS HIGH (A8-A15) OR LOW (A0-A7) BYTES			
RAM	ALL SINGLE AND MANY MULTIPLE FAULTS IN DATA AND DATA PARITY BITS	DATA PARITY	C9D etc.	READ
	ALL FAULTS IN ADDRESS PARITY BITS	ADDRESS PARITY	C11	READ
	ANY FAULT CAUSING THE EFFECTIVE INTERNAL ADDRESS TO BE AN ODD HAMMING DISTANCE FROM THE DESIRED ADDRESS IN ITS HIGH (A8-A15) OR LOW (A0-A7) BYTES			
BUFFER B5	ALL INTERNAL AND OUTPUT FAULTS	ADDRESS PARITY	C10*	READ
BUFFER B6	ALL INTERNAL AND OUTPUT FAULTS	ADDRESS PARITY	C11*	READ/ WRITE

Notes : 1) Faults detected exclude pin stuck-at faults, except where stated.
2) An address parity error is an error in parity bits AP1 and/or AP2.
3) The checkers are those in Fig. 9.24.
4) * indicates that a fault may be detected when the CPUs are halted.
5) C9D is the data bus checker of C9.
6) Read = Read from that device.
7) Read/Write = read/write from/to any device.

FIGURE 9.27 MEMORY FAULTS

NORMAL OPERATION

CHECKER INDICATIONS						SINGLE FAULTS DETECTED (DATA BUS ONLY, INCLUDING PARITY)
C6D	C7D	C8	C9D	C13	C14D	
	X	X	X			B3b SA0/1 ($\xrightarrow{M}$),RB S/C ($\xrightarrow{M}$),B4a SA0 ($\xrightarrow{M}$)
	X			X	X	B3b SA0/1 ($\xrightarrow{I}$),RB S/C ($\xrightarrow{I}$),B7a SA0 ($\xrightarrow{I}$)
X	X					B3b SA0/1 ($\xleftarrow{M}\xleftarrow{I}$), D1 O/C ($\xleftarrow{M}$), D2 O/C ($\xleftarrow{I}$),RB S/C ($\xleftarrow{M}\xleftarrow{I}$)
X	X	X				B4a SA0 ($\xleftarrow{M}\xleftarrow{I}$),B4a SA1 ($\xleftarrow{M}$),RBM S/C ($\xleftarrow{M}$)
X	X			X		B7a SA0 ($\xleftarrow{M}\xleftarrow{I}$),B7a SA1 ($\xleftarrow{I}$),RBI S/C ($\xleftarrow{I}$)
		X	X			T1BE S/C or O/C ($\xrightarrow{M}$),T1CE O/C ($\xrightarrow{M}$),T1BC O/C ($\xrightarrow{M}$),BD1 SA0 ($\xrightarrow{M}$),RB1 O/C ($\xrightarrow{M}$),B4a SA1 ($\xrightarrow{M}$),RBM S/C ($\xrightarrow{M}$)
		X				T1CE S/C ($\xleftarrow{I}\xrightarrow{I}$),BD1 SA1 ($\xleftarrow{I}\xrightarrow{I}$),D1 S/C ($\xleftarrow{I}\xrightarrow{I}$),T1BC S/C ($\xleftarrow{I}\xrightarrow{I}$)
				X	X	T2BE S/C or O/C ($\xrightarrow{I}$),T2CE O/C ($\xrightarrow{I}$),T2BC O/C ($\xrightarrow{I}$),BD2 SA0 ($\xrightarrow{I}$),RB2 O/C ($\xrightarrow{I}$),B7a SA1 ($\xrightarrow{I}$),RBI S/C ($\xrightarrow{I}$)
				X		T2CE S/C ($\xleftarrow{M}\xrightarrow{M}$),BD2 SA1 ($\xleftarrow{M}\xrightarrow{M}$),D2 S/C ($\xleftarrow{M}\xrightarrow{M}$),T2BC S/C ($\xleftarrow{M}\xrightarrow{M}$)
	X	X				B4a SA0 ($\xrightarrow{I}$)
	X			X		B7a SA0 ($\xrightarrow{M}$)

CPUs HALTED

X	X					B3b SA0
X	X	X				B4a SA0
X	X			X		B7a SA0

NOTES : 1) X = fault indicated.
2) SA0 = stuck-at-0 ; SA1 = stuck-at-1.
3) O/C = open circuit ; S/C = short circuit.
4) $\xleftarrow{M}$ = read from memory board ; $\xrightarrow{M}$ = write to memory board.
5) $\xleftarrow{I}$ = read from I/O board ; $\xrightarrow{I}$ = write to I/O board.
6) T1,RB1,D1,BD1 = any of these components in the bidirectional isolators of I1.
7) T2,RB2,D2,BD2 = any of these components in the bidirectional isolators of I2.
8) CnnD is the data bus checker of Cnn.
9) Checkers, buffers, isolators and bus tie-up resistors are those of Fig. 9.24.
10) Transistors, resistors, diodes and base drive buffers are those of Fig. B.5.

FIGURE 9.28 BIDIRECTIONAL ISOLATOR FAULTS

NORMAL OPERATION			
CHECKER INDICATIONS			SINGLE FAULTS DETECTED
C7^	C9^	C14^	
X	X	X	B3b SA0/1,RB S/C
	X		B4a SA0/1, ALL T1 FAILURES, RB1 O/C, RBE1 S/C, RC1 S/C
		X	B7a SA0/1, ALL T2 FAILURES, RB2 O/C, RBE2 S/C, RC2 S/C
X		X	RB1 S/C
X	X		RB2 S/C

CPUs HALTED : ADDRESS BUSES 1 AND 2			
X	X	X	B3b SA0
	X		B4a SA1#
		X	B7a SA1#

CPUs HALTED : CONTROL BUS			
X	X	X	ø2B SA0 , VMAB SA1, R/$\overline{W}$B SA0, $\overline{RST}$B SA0, CPB SA1
	X		ø2BM SA0 , VMABM SA1, R/$\overline{W}$BM SA0, $\overline{RST}$BM SA0, CPBM SA1
		X	ø2BI SA0 , VMABI SA1, R/$\overline{W}$BI SA0, $\overline{RST}$BI SA0, CPBI SA1

NOTES : 1) X = fault indicated.
2) SA0 = stuck-at-0 ; SA1 = stuck-at-1.
3) O/C = open circuit ; S/C = short circuit.
4) T1, RB1, RBE1, RC1 = any of these components in the unidirectional isolators of I1.
5) T2, RB2, RBE2, RC2 = any of these components in the unidirectional isolators of I2.
6) BM = between I1 and B4.
7) BI = between I2 and B7.
8) ^ = A1 for address bus 1 (A0-A7,AP1), A2 for address bus 2 (A8-A15,AP2) or C for the control bus (ø2,VMA,R/$\overline{W}$,$\overline{RST}$,CP).
9) Checkers, buffers and isolators are those of Fig. 9.24.
10) Transistors and resistors are those of Fig. B.9.
11) # = includes line between buffer and isolator.

FIGURE 9.29 UNIDIRECTIONAL ISOLATOR FAULTS

CHAPTER TEN : CONCLUSIONS AND FURTHER WORK

The purpose of this investigation has been twofold. Firstly, to devise a method for the design of totally self checking (TSC) circuits from first principles and, secondly, to develop a microprocessor based system with on line fault detection, so that faults can be diagnosed to individual integrated circuits (ICs), without the need for external equipment. These two objectives have been brought together in the experimental self checking computer of Chapter 9.

The first objective of the investigation has been achieved with a technique based on the testing requirements of single and cascaded logic gates. A TSC circuit is, by definition, self testing and fault secure. Fault security is automatic in a circuit where each output of the 1-out-of-2 encoded pair is computed with an independent sub-circuit. Therefore it is only necessary to determine if the circuit is self testing, for it to be TSC. This is achieved in the case of 2-level AND/OR and OR/AND structures with the method proposed in Chapter 6. The Boolean function of the circuit depicted on a Karnaugh map is visually inspected to determine if the circuit is self testing. This is a rapid process, which will identify why the circuit is not self testing, so that it can be modified and re-evaluated.

The technique of using Karnaugh maps to design TSC circuits is not suitable for TSC circuits with a large number of inputs. However, complex TSC circuits or networks are generally constructed from a number of smaller TSC circuits, either by design or necessity. Examples are the TSC comparator and TSC 1-out-of-n code checker respectively. The design of other TSC circuits, including m-out-of-n and Berger code checkers, should be attempted using the Karnaugh map method. The technique does not cater for TSC circuits constructed from EXOR gates, notably parity checkers, which have been largely

designed by intuition. This is an another area which could be explored.

The second objective of the investigation has been achieved by the application of a number of techniques to convert a minimal, but typical, microprocessor based system from being unchecked to fully checked. These techniques include the parity encoding of all data transmission paths, the duplication of VLSI devices and control circuitry, the storage of address parity bits in memory and the installation of signal isolating circuits. Checking circuits are then added at strategic points in the system to achieve the desired level of fault diagnosis from their output indications. These checking circuits check codes, compare two sets of inputs and check periodic signals. Additional circuitry is provided to display and process their outputs.

The major limitation of the checking circuits, is that if they are designed strictly according to the Boolean function of their specification, then faults within them can mask the indication of a fault in the system they are monitoring. This problem is resolved by designing TSC checkers. This then brings together the two original objectives. The result is an experimental self checking computer for the demonstration and evaluation of both objectives.

The two principal objectives of the experimental computer have also been met. In the areas which are checked, faults in an IC or its associated components, such as isolating circuits, are diagnosable to that chip. If only board level fault diagnosis is necessary, then considerably less fault indication and error control logic will be required.

The implementation of various self checking techniques has also been demonstrated. These are duplication and comparison, parity generation and checking, 1-out-of-n code

checking and periodic signal checking. It has been shown that duplication and comparison along with parity encoding are the two most widely used techniques.

The isolating circuits enhance the fault diagnosis of the system, rather than assisting its self checking abiltites. Overall, a considerable amount of additional hardware is required, primarily for the self checking techniques, but none of it is complex.

All types of single fault have been simulated in the isolating circuits and many single and multiple faults in other checked parts of the system. If the efficiency of the fault detection mechanisms is to be evaluated for all faults (single and multiple), then a computer simulation of faults in the system will be more appropriate than a physical fault simulation.

Additional checking mechanisms could be added to the experimental computer to make it totally self checking. Many more isolation circuits would then be required, resulting in a considerable increase in the hardware for checking. So far, there have been no constraints placed on the amount of hardware used for checking purposes. However, an examination of the checked computer reveals that it already has considerably more than five times the number of IC packages used in the original unchecked system. Clearly, such a system will only be economically viable if ICs are available which are self checking versions of existing devices, or provide a set of checking mechanisms for use with standard devices, as indicated in Chapter 8.

If a single chip or group of chips can be identified as faulty from the indications of the checking circuits, which might require the combined indications from both a halted and an operational system, then it would be possible to display this information, in words, on a terminal. This would extend the fault diagnosis capabilities of the

system from a manual to an automatic process and allow a rapid replacement of the faulty chip(s). Alternatively, this information could be used in fault tolerant applications to effect error correction and/or error recovery. In all cases it must be remembered that the checking circuits are TSC, so that an indicated error may be due to a fault in the checker itself and not a fault in the signals it is monitoring.

Proposals have been given in Chapter 9 for the construction of a self checking MC6800 microprocessor and a self checking MC6850 ACIA. Both schemes use the duplication and comparsion technique with the addition of a tri-state buffer. The MC6821 PIA can be checked in a similar manner to the ACIA, for its application in the experimental computer. However, it is more difficult to construct a general self checking PIA, because, externally, it is not known which I/O lines have been programmed as inputs and which as outputs. Access must be gained to the internal data direction and control registers for this information, or a redundant set of registers added externally. A self checking PIA needs to be the subject of a separate investigation.

The isolation circuits should be further developed to remove existing limitations in their design. Certain undetectable faults ideally need to be eliminated. In addition, the circuits rely on there being more than one data path, ie. a structure with a single transmitter and greater than two receivers, for the detection of certain faults. If the isolators are installed in single transmitter to single receiver structures, then a number of additional faults become undetectable.

After further development, the isolating circuits can then be integrated as a separate package, or within the devices they are isolating. The latter is achieved by modifying the input and output circuitry of a gate, so that a physical stuck-at fault at an IC pin is impossible; i.e.

an input or output line of a gate can still be driven to a logic 0 or 1 in the presence of an internal input or output 'stuck-at' fault. However, whilst unswitched receiver input isolators would assist fault diagnosis in an unswitched single tranmitter to multiple receiver structure, switched isolators in all switched structures would not improve fault diagnosis, without the additional checkers between each transmitter output or receiver input and its isolating circuit. Some further investigation is therefore required into this aspect of isolation circuit integration.

The use of opto-couplers and transformers as isolating circuits was considered during the development of these circuits. Opto-couplers were rejected on the basis of their cost, a relatively high input current and a large propagation delay, whilst a commercially available transformer bus isolator package was rejected because it would not process a 1MHz clock waveform. However, both components, in principle, are ideal as isolation circuits, particularly opto-couplers which are easily integrated, and should therefore be investigated further.

In most of the discussion about self checking checkers, only TSC checkers have been considered. The exception to this is the 1-out-of-n code checker used in the experimental computer, which, in general, is a self testing only checker. However for its specific application to check the outputs of a 3 to 8 line decoder, in which the effects on these outputs of all single faults have been ascertained, it is deemed to be TSC. Self testing only (STO) and partially self checking (PSC) checkers could have been employed in the experimental computer, since the immediate detection of errors is not essential, only the eventual detection of errors; i.e. there has been more of an emphasis on self testing rather than fault security. However, it is considered more important to develop and evaluate checkers suitable for any application, hence the bias towards TSC checkers. The additional hardware

required for TSC checkers is unlikely to be a problem when they are integrated.

In Chapter 6 the process of level merging was presented for TSC equality checkers constructed from cascaded 2-input Morphic AND gates. The technique, in general, reduces the circuitry required for these checkers and also their propagation delay. Its application is, however, dependent on the size of the comaprator. Whilst it is ideal for comparators implemented with discrete logic, as in the experimental computer, it is more likely than the 2-level AND/OR or OR/AND structures, also presented in Chapter 6, will be adopted for integrated TSC comparators with less than nine input pairs.

The theory and design of TSC sequential circuits has not been extended from that given in Chapter 5, principally because there was no requirement for them in the experimental computer. The most likely application of self checking sequential circuits in such a system, aside from any sequential control circuitry, is a multiple phase clock generator. It would therefore be worth investigating the possibility of TSC sequential circuit design using a method based on Karnaugh maps.

It would take some considerable time to create a complete set of self checking versions of standard integrated circuits. In addition, the amount of error processing required in a system using self checking chips exclusively, where each chip has a 1-out-of-2 encoded output error signal, would be vast. The fault coverage of such a system would also have to be carefully analysed.

Overall, at present, it is considered more beneficial to pursue the development and construction of integrated circuits containing a set of self checking mechanisms, including self checking checkers, which will allow self checking to be incorporated into a system using standard integrated circuits. The construction and development of

isolating circuit packages should also be pursued, so that they and self checking checkers can be positioned to provide adequate fault diagnostic information. A microprocessor based system can then be designed with a minimum hardware overhead to provide the desired level of self checking and fault diagnosis.

APPENDIX A : PRACTICAL IN-CIRCUIT EMULATION

Extensive in-circuit emulation has been practically implemented on the microprocessor based system depicted in Fig. A.1, using a Millenium Microsystem Analyser (in-circuit emulator). This work is detailed below.

A.1 : OPERATION OF THE PROCESSOR SYSTEM

Refering to Fig. A.1, a terminal connected to the 'terminal' programmable communications interface (PCI), via RS232 or 20mA interfaces, can be effectively switched through the 'processor' PCI and further RS232 or 20mA interfaces to any one of sixteen processing units (computers). The communication path between the terminal and its processor is 'full duplex'.

The processor is selected by a four bit binary code stored in the route latch which controls a multiplexer/demultiplexer configuration, connecting the appropriate processor to the processor PCI. This code is indicated (in hexadecimal) on the (7-segement) route display.

Not all processors are available to the terminal, this being controlled by the 'allowed processor' switches. In addition, one of seven possible speeds of communication is selected by the 'baud rate selection' switch.

Each system has four terminals, so the circuitry shown in the dotted box in Fig. A.1 is duplicated three times. Note that the terminal and processor PCIs operate in pairs.

A terminal connected via 20mA interfaces to the 'terminal' PCI, provides monitoring and control for the whole system.

A.2 : THE TESTS

A set of seven programs was written to test the

system, using the operating system of the Microsystem Analyser [A.1,A.2,A.3]. These reside in two EPROMs located on the analyser. Prompt and fault messages are displayed on the twenty character display of the instrument. The tests run individually, or in sequence under emulator control and consist of the following:

1) Switch Test:

   a) The emulator prompts for the 'allowed processor' and 'baud rate selection' switches to be set to 55_{16} and then reads them to check this. An error message gives the switch number and the actual data read.

   b) As for a), but switches set to AA_{16}.

2) EPROM Test.

   A checksum is compiled and verified for each EPROM in turn. An error message gives the actual checksum compiled.

3) RAM Test.

   An 8-bit binary number, which starts at 00_{16}, is Exclusively ORed with the most significant (MS) address byte of the first location in memory and the result stored in that location. The binary number is then incremented, Exclusively Ored with the MS address byte of the second location and stored in that location. This procedure continues until all the RAM has been written to. RAM contents are then read back and compared with that written. An error message details written and read data.

4) Display Test.

   0, 1, 2 through to F_{16} is written to route display '0' and then repeated for displays 1, 2 and 3 in succession. A software delay is incorporated between changes, so that the sequence can be observed. This is a purely visual test.

5) PCI Test 1.

Links are made between the processor PCI interfaces and the terminal PCI interfaces, as detailed in Fig. A.2a. For each connection indicated, a complete binary count ($00\text{-}FF_{16}$) is transmitted at all possible baud rates (110-19200 baud) from the processor PCI to the terminal PCI and then vice versa. Interrupts normally used to indicate 'transmitter (TX) ready' and 'receiver (RX) ready' are inhibited. The error message details the two PCIs in use, the processor port selected, plus the direction and baud rate of communication, along with one of three faults:

a) No character transmitted.
b) No character received.
c) Character sent and character received (when different).

6) PCI Test 2.

This is the same as for PCI test 1, except it uses a different set of connections, as detailed in Fig. A.2b, and also includes tests using the monitor PCI.

7) Interrupt Test.

Using the same links as for PCI test 2, this test detects the occurrence of transmitter and receiver interrupts for all PCIs, by transmitting 55_{16} at all baud rates and in both directions; terminal to processor and processor to terminal. The interrupts are interrupt requests (IRQ) for the terminal and processor PCIs and a non-maskable interrupt (NMI) for the monitor PCI. An EPROM contains software routines and vectors to handle the interrupts, replacing the upper control EPROM of the board. A prioritisation of interrupts occurs for the IRQ and this is also tested. An error message for the general interrupt test details the two PCIs in use, the processor port selected, plus the direction and baud rate of communication and one of four errors:

a) No IRQ detected.
b) Correct and actual levels of IRQ priority.
c) No NMI detected.
d) IRQ level detected instead of NMI.

The error message for the interrupt priority test gives either a) or b) above.

A.3 : THE TESTS IN USE

The tests have proved to be an effective means of testing and fault diagnosing the board, in what is essentially an automated procedure, although the emulator halts on errors. They also highlight the main limitation of in-circuit emulation and for that matter functional testing, which is circuit visibility, or rather a lack of it.

Consider the following example. In test 5) a PCI error is detected with a fault message of 'NO RX', i.e. the receiving PCI has not detected a character. Referring to Fig. A.3, it cannot be ascertained directly if the fault lies within the:

i) Transmitting PCI - eg. theoretical but not physical transmission.
ii) Transmitter Interface - eg. component failure.
iii) Interface Links - eg. not properly connected.
iv) Receiver Interface - eg. component failure.
v) Receiving PCI - eg. physical but not theore-tical reception.

It is assumed that the main data, address and control buses are functional, since this would have been determined from previous tests.

The fault is located by running another test routine, which continuously transmits a character at a predetermined baud rate between a definable pair of PCIs. An oscilloscope is then used to trace signal flow between the two PCIs.

A.4 : REFERENCES

A.1 Millenium Microsystem Analyser; Operators Manual; Pub. No. 87000001, release 2.0 ;May 1980; Millenium Systems Inc.

A.2 Programming with uSA Microsystem Analyser - B. Hordos; uSA applications note no. 1; Millenium Systems Inc.

A.3 Diagnostic programming for microprocessor-based systems - B. Hordos; uSA applications note no. 2; Systems Inc.

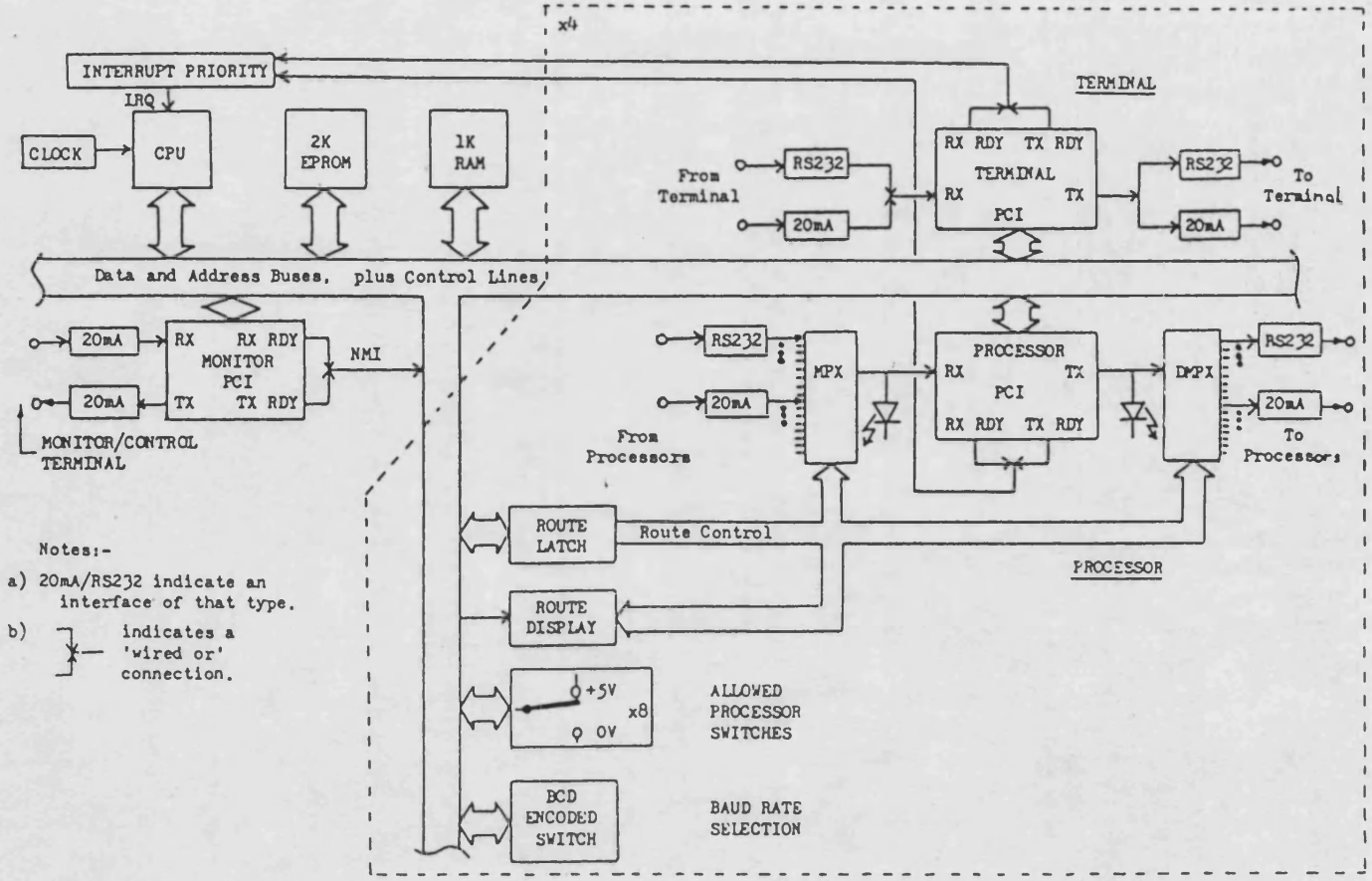


FIGURE A.1 MICROPROCESSOR BASED SYSTEM USED WITH IN-CIRCUIT EMULATION

TERMINAL PCI	(No.)	0		1		2		3	
CONNECTED VIA INTERFACE	(Type)	RS232	20mA	RS232	20mA	RS232	20mA	RS232	20mA
TO PROCESSOR PORT	(No.)	4	C	5	D	6	E	7	F
FOR SELECTION BY PROCESSOR PCI	(No.)	4		5		6		7	

(a) CONNECTIONS FOR PCI TEST 1

TERMINAL PCI	(No.)	0		1		2		3		8
CONNECTED VIA INTERFACE	(Type)	RS232	20mA	RS232	20mA	RS232	20mA	RS232	20mA	20mA
TO PROCESSOR PORT	(No.)	0	8	1	9	2	A	3	B	F
FOR SELECTION BY PROCESSOR PCI	(No.)	4		5		6		7		4

(b) CONNECTIONS FOR PCI TEST 2

FIGURE A.2 PCI TEST CONNECTIONS

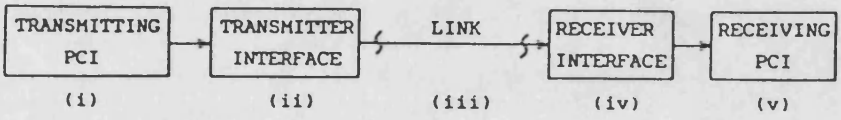


FIGURE A.3 PCI TEST EXAMPLE

APPENDIX B : SIGNAL ISOLATION CIRCUITS

B.1 : INTRODUCTION

Sections 7.8.2 and 8.2.8 have demonstrated a need for signal isolation circuits. These sections have also discussed the positioning of isolators in various circuit structures. This appendix summarises the work carried out to design bidirectional and unidirectional isolating circuits, principally for the self checking computer of Chapter 9.

Isolating circuits are allowed to fail, but not in either of the following two ways:

1) A stuck-at fault at the signal terminal connected to the line from which it is providing isolation.
2) A undetectable short between the two signal terminals.

Both of these failures destroy the isolation properties of the circuit. The effects of single failures appropriate to each component used form the basis of fault tables for the various isolation circuits. These are mainly short and open circuits in resistors, diodes and transistor junctions. All of these failures are based on discrete components. Faults caused by substrate failures, (as opposed to circuit failures) when the components are integrated, are not considered.

The structure used to evaluate each isolating circuit is one of the following:

1) Single transmitter to two receivers.
2) Two transmitters to a single receiver.
3) Master transceiver to two slave transceivers.

They are all assumed to represent one bit of a k-bit parity encoded bus.

B.2 : SWITCHED UNIDIRECTIONAL ISOLATORS

The unidirectional isolating circuits used by Moreira de Souza et al [B.1] in their research computer have already been presented in section 7.8.2. The isolator they used for single transmitter to multiple receiver structures, a series resistor and CMOS buffer, is shown again in Fig. B.1a. Note that the receivers are assumed to be high speed CMOS buffers [B.2], so the series CMOS buffer is now unnecessary. It is also assumed that only one path, out of the two indicated, is selected for data transmission at any time. Fig. B.1b gives the fault table for this structure. This reveals, as expected, that a short circuit isolating resistor is undetectable. Moreira de Souza et al overcome this problem by using wirewound resistors, which have a small probability of short circuit failures. However, wirewound resistors are expensive and difficult to integrate, so an alternative solution is required. The fault table in Fig. B.1b also shows that certain faults will be indicated when no data path is selected.

If the resistors in Fig. B.1a are replaced by diodes, then the circuit and fault table become those in Fig. B.2. Some faults now create an indeterminate condition. This occurs when the anode of a diode is stuck-at 1 (SA1) and a 0 is applied to its cathode. The actual voltage levels at either end of the diode will depend on the forward volt drop across it, V_f, as well as the source resistances of the SA1 fault and driving 0. For normal use, V_f should be as low as possible, so that a transmitted 0 is still a 0 at each receiver ('0' + V_f). Germanium OA47 diodes have been used for practical purposes [B.3], but modern schottky signal diodes are also suitable and readily integrated [B.4].

The fault table in Fig. B.2b also shows that a short circuit diode is, again, not detectable. This situation can be resolved, however, by replacing the diodes with

switched diodes, i.e. transistors. Each transistor is turned on when the data flow is through its associated buffer. The circuit and fault table for this configuration are given in Fig. B.3. The transistor and its base resistor are selected on the basis of voltage and current levels, propagation delay and rise times. In the prototype, these were a Texas 2N2926A transistor [B.5] and a 1k ohm resistor. From Fig. B.3b, there are a number of indeterminate faults. A short circuit base resistor gives the condition mentioned above. In this instance, it is across the base-emitter junction of the transistor. This condition, however, will only occur when that transistor is active. A receiver input SA1 also results in a similar condition, this time between the collector and the emitter of the transistor, but again, only when that transistor is active.

The other isolation circuit used by Moreira de Souza et al is a series diode for single transmitter to multiple receiver structures. This is shown again in Fig. B.4a. Fig. B.4b gives the fault table for this structure, which, once again, has the indeterminate condition described above. All other faults, though, are definitely detectable. Replacing the diodes with transistors is therefore unnecessary. If they were, a base drive buffer output SA1 fault, i.e. the transistor permanently turned on (see Fig. B.3a), is not detectable in any case.

B.3 : BIDIRECTIONAL ISOLATORS

The experimental self checking computer, described in Chapter 9, has a bidirectional data bus. Moreira de Souza et al do not consider isolators for such a bus [B.1]. Their memory and peripherals have separate data in and data out lines, so the data bus consists of two unidirectional buses. The data bus of the experimental computer could be split into two undirectional buses for isolation purposes, but this would then not represent a conventional system.

A master-slave transceiver bus structure is formed by the merging of the structures in Figs. B.3a and B.4a. The isolation circuits used in the structures of Figs. B.3a and B.4a. are also merged to form an isolating circuit for the master-slave structure. Then the overall circuit and fault table become those in Fig. B.5. Another type of indeterminate condition is now introduced. This occurs when the cathode of a diode is stuck-at-0 (SA0) and a 1 is applied to its anode.

In Fig. B.6 a full fault analysis is performed on the fault table of Fig. B.5b. Indeterminate conditions have been resolved by practical experiment; i.e. checker indications represent the most likely effect of faults which create inderminate conditions. A short circuit base resistor is the only fault, in general, which is undetectable. However, the buffer which supplies the transistor base current may be doing so for more than one isolating circuit. Depending on the drive capability of this buffer and the value of the base resistors, then a short circuit base resistor may sufficiently lower the output of the buffer, so that some or all of the other transistors turn off, in which case the fault will be detected. The addition of a base-emitter resistor, as used in the next section, is resisted, since it would make the isolating circuits more complex.

For three fault indications identified in Fig. B.6, the fault cannot be diagnosed to one of the three buffers. Note that isolators are considered to be part of their associated buffer. However, the fault indications for no system activity, see Fig. B.6, resolve this problem.

B.4 : UNSWITCHED UNIDIRECTIONAL ISOLATORS

All the unidirectional and bidirectional structures considered so far have assumed the data path to be switched. This assumption is not satisfactory for address and

control buses, in particular, where all buffers are permanently enabled. The single transmitter to multiple receiver structure is the most appropriate for these unidirectional buses. An alternative isolator to the switched transistor is therefore required.

The first circuit evaluated for this purpose, was that shown in Fig. B.7. It is similar to the input circuitry of a standard TTL logic gate [B.6]. However, it is unusable, because two input faults are undetectable. These are base-emitter and collector-emitter shorts in transistor T1. Various modifications were made to the circuit to detect these faults. However, the modifications then created a significant number of undetectable faults in other parts of the circuit.

A conventional single stage transistor amplifier was then considered, as shown in Fig. B.8a. Fig. B.8b gives the fault table for this circuit. Note that a checker is not required at the input of any receiver. Since the buffers are permanently enabled, any fault indication at a receiver input will also be indicated at its output.

The fault table in Fig. B.8b reveals that a short circuit input resistor is not detected. This assumes that neither the transmitter or the isolating transistor is destroyed. The ideal solution is for a short circuit input resistor to sufficiently drag down the output level of the transmitter, such as to turn off the transistor in all other isolating circuits connected to this node. This does not happen in the circuit of Fig. B.8a, because the input level, under these conditions, is still higher than the level at which the transistor turns off, referred to as the trigger level. A transistor will turn off under these conditions, though, if the resistance of its input resistor is increased to greater than 10k ohms. However, the propagation delay of the isolator is then unacceptably increased. An alternative solution is to add a base-emitter resistor to the circuit of Fig. B.8a. This will

divert current away from the base of the transistor, thereby increasing the trigger input level. A value of 820 ohms was selected in the prototype for this purpose, on the basis of trigger level and propagation delay.

Fig. B.9 shows the modified structure and its fault table. Unfortunately, open circuit base-emitter and collector resistors are both, in general, undetectable. An open circuit collector resistor is equivalent to an open circuit tie-up resistor. During its high impedance state, a bus line with an open circuit tie-up resistor will float. In this state it is possible for the line to look like a 0. If so, the fault will be detected.

If a base-emitter resistor is open circuit, then undetectable second faults, excluding collector resistor faults, are a short circuit base resistor and an open circuit base-emitter resistor in the other isolating circuit. The probability of either of these faults occurring as second faults, is small. The probability of detecting a second fault, as well as the probability of the first fault not being an open circuit base-emitter resistor, are both high, typically greater than 90%.

In Fig. B.10 a full fault analysis is performed on the fault table of B.8b. All indicated faults are diagnosable to their source, one of the three buffers.

B.5 : REFERENCES

B.1) A research oriented microcomputer with built in auto-diagnostics - J. Moreira de Souza, E. Peixoto Paz, C. Landrault; FTCS-6*; pp. 3-8.

B.2) High-speed CMOS replacements for LS TTL promise continued viability of jellybean logic - R. H. Cushman; EDN; March 17, 1983; pp. 64-74.

B.3) Mullard Technical Handbook, Book One, Semiconductor Devices, Part 3, Diodes; Mullard Ltd.; London, England; September 1977.

B.4) 1A, 3A Schottky Rectifier Diodes; 11DQ, 31DQ series; Bulletin E2126; International Rectifier; Surrey, England.

B.5) Texas Instruments Semiconductor Components Data Book Four, Transistors; Texas Instruments Ltd; Bedford, England; 1971.

B.6) The TTL book for design engineers - The engineering staff of Texas Instruments components group; Texas Instruments; Fourth European edition; 1980.

* see below.

FTCS-1 : The 1st Annual International Symposium on Fault-Tolerant Computing; Pasadena,California,USA; March 1971.

FTCS-2 : The 2nd Annual International Symposium on Fault-Tolerant Computing; Boston, Massachusetts, USA; June 1972; IEEE Pub. No. 72 CH0623-4C.

FTCS-3 : The 3rd Annual International Symposium on Fault-Tolerant Computing; Palo Alto, California, USA; June 1973; IEEE Pub. No. 73 CH0772-4C.

FTCS-4 : The 4th Annual International Conference on Fault-Tolerant Computing; Urbana, Illinois, USA; June 1974; IEEE Pub. No. 74 CH0864-9C.

FTCS-5 : The 5th Annual International Symposium on Fault-Tolerant Computing; Paris France; June 1975; IEEE Pub. No. 75 CH0974-6C.

FTCS-6 : The 6th Annual International Symposium on Fault-Tolerant Computing; Pittsburgh, Pennsylvania, USA; June 1976; IEEE Pub. No. 76 CH1094-2C.

FTCS-7 : The 7th Annual International Conference on Fault-Tolerant Computing; Los Angeles, California, USA; June 1977; IEEE Pub. No. 77 CH1223-7C.

FTCS-8 : The 8th Annual International Conference on Fault-Tolerant Computing; Toulouse, France; June 1978; IEEE Pub. No. 78 CH1286-4C.

FTCS-9 : The 9th Annual International Symposium on Fault-Tolerant Computing; Madison, Wisconsin, USA; June 1979; IEEE Pub. No. 79 CH1396-1C.

FTCS-10 : The 10th Annual International Symposium on Fault-Tolerant Computing; Kyoto, Japan; October 1980; IEEE Pub. No. 80 CH1604-8.

FTCS-11 : The 11th Annual International Symposium on Fault-Tolerant Computing; Portland, Maine, USA; June 1981; IEEE Pub. No. 81 CH1600-6.

FTCS-12 : The 12th Annual International Symposium on Fault-Tolerant Computing; Santa Monica, California, USA; June 1982; IEEE Pub. No. 82 CH1760-8.

FTCS-13 : The 13th Annual International Symposium on Fault-Tolerant Computing; Milan Italy; June 1983; IEEE Pub. No. 83 CH1894-5.

FTCS-14 : The 14th Annual International Conference on Fault-Tolerant Computing; Kissimmee, Florida, USA; June 1984; IEEE Pub. No. 84 CH2050-3.

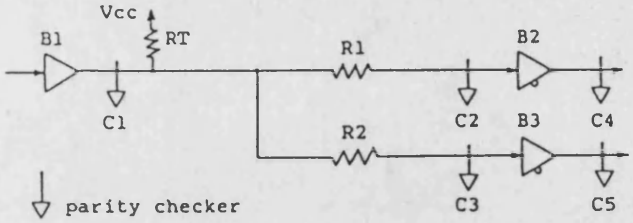
FTCS-15 : The 15th Annual International Symposium on Fault-Tolerant Computing; Ann Arbor, Michigan, USA; June 1985; IEEE Pub. No. 85 CH2143-6

1979 Test Conf. : 1979 IEEE Test Conference; Cherry Hill, New Jersey, USA; October 1979; IEEE Pub. No. 79 CH1509-9C.

1980 Test Conf. : 1980 IEEE Test Conference; Philadelphia, Pasadena, USA; 11-13 November 1980; IEEE New York 1980.

Autotestcon '80 : IEEE Autotestcon '80; Washington, DC, USA; 2-5 November 1980; IEEE New York 1980.

Infotech : Infotech State-of-the-Art Report on Computer System Reliabilty, Series 3; Infotech Information Ltd., Maidenhead, England; 1975.

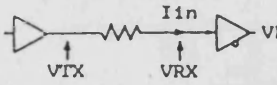


(a) CIRCUIT

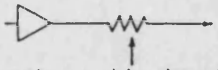
FAULT	B1 TX TO B2					B1 TX TO B3				
	C1	C2	C3	C4	C5	C1	C2	C3	C4	C5
B1 O/P SA0	X*	X*	X*	X		X*	X*	X*		X
B1 O/P SA1	X	X	X	X		X	X	X		X
B2 I/P SA0		X*		X			X*			
B2 I/P SA1		X		X			X			
B3 I/P SA0			X*					X*		X
B3 I/P SA1			X					X		X
R1 O/C		X		X			X			
R1 S/C										
R2 O/C			X					X		X
R2 S/C										
RT O/C										
RT S/C	X	X	X	X		X	X	X		X

(b) FAULT TABLE

Notes : 1) B2/B3 enabled when transmission is via that buffer.
2) X = fault indicated.
3) X* = fault indicated with no buffer enabled.
4)



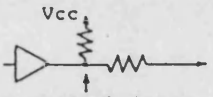
5)



Can still drive line regardless of fault.

acts as tie-down/up resistor

but:

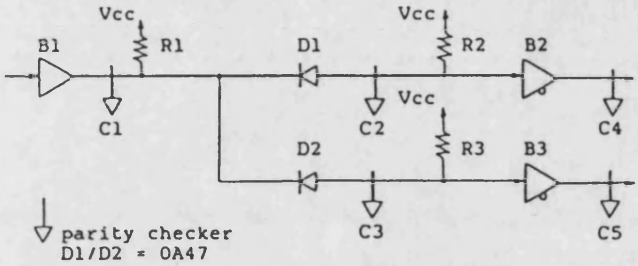


potential divider action
(assumed still at '1' in table above)

6) Tie-up resistors not required at C2/C3. In fact they would be detrimental to circuit operation.

7) R1 S/C, R2 S/C and possibly RT O/C not detected.

FIGURE B.1 RESISTORS AS UNIDIRECTIONAL ISOLATORS



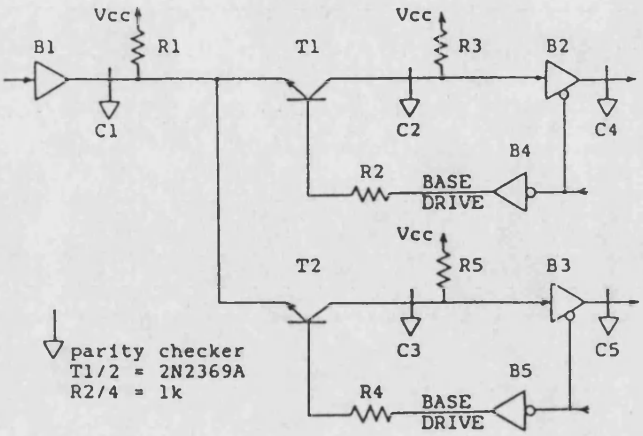
(a) CIRCUIT

FAULT	B1 TX TO B2					B1 TX TO B3				
	C1	C2	C3	C4	C5	C1	C2	C3	C4	C5
B1 O/P SA0	X^	X^	X^	X		X^	X^	X^		X
B1 O/P SA1	X	X	X	X		X	X	X		X
B2 I/P SA0		X^		X			X^			
B2 I/P SA1	#	#	#	#		#	#	#		#
B3 I/P SA0			X^					X^		X
B3 I/P SA1	#	#	#	#		#	#	#		#
D1 O/C		X		X			X			
D1 S/C										
D2 O/C			X					X		X
D2 S/C										

(b) FAULT TABLE

Notes : 1) B2/B3 enabled when transmission is via that buffer.
2) X = fault indicated.
3) X^ = fault indicated with no buffer enabled.
4) # = indeterminate fault : '0'—▷|— SA1
5) R1 and R2 keep B2 and B3 inputs at 1 when B1 outputs a 1.
6) R1 S/C ≡ B1 O/P SA1 } Any of these resistors open
R2 S/C ≡ B2 I/P SA1 } circuit results in a floating
R3 S/C ≡ B3 I/P SA1 } input and a possible fault.
7) D1 S/C and D2 S/C are not detected.

FIGURE B.2 DIODES AS UNIDIRECTIONAL ISOLATORS - 1



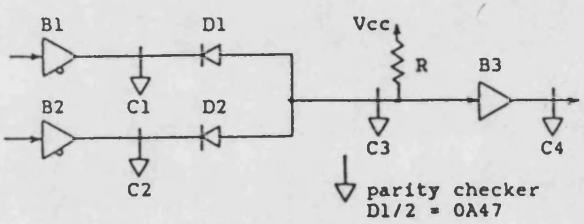
(a) CIRCUIT

FAULT	B1 TX TO B2					B1 TX TO B3				
	C1	C2	C3	C4	C5	C1	C2	C3	C4	C5
B1 O/P SA0	X^	X		X		X^		X		X
B1 O/P SA1	X	X		X		X		X		X
B2 I/P SA0		X^		X			X^			
B2 I/P SA1	#	#		#						
B3 I/P SA0			X^					X^		X
B3 I/P SA1						#		#		#
B4 O/P SA0		X		X						
B4 O/P SA1							X			
B5 O/P SA0								X		X
B5 O/P SA1			X							
T1BE O/C		X		X						
T1BE S/C		X		X						
T1BC O/C		X		X						
T1BC S/C							X			
T1CE O/C		X		X						
T1CE S/C							X			
T2BE O/C								X		X
T2BE S/C								X		X
T2BC O/C								X		X
T2BC S/C			X							
T2CE O/C								X		X
T2CE S/C			X							
R2 O/C		X		X						
R2 S/C	#	#		#						
R4 O/C								X		X
R4 S/C						#		#		#

(b) FAULT TABLE

Notes : 1) B2/B3 enabled when transmission is via that buffer.
2) X = fault indicated.
3) X^ = fault indicated with no buffer enabled.
4) # = indeterminate fault : '0' —▷|— SA1
5) Transistor BC S/C provides a diode (BE) as required, except that it is permanently on.
6) R1 S/C = B1 O/P SA1 } Any of these resistors open
R3 S/C = B2 I/P SA1 } circuit results in a floating
R5 S/C = B3 I/P SA1 } input and a possible fault.

FIGURE B.3 TRANSISTORS AS UNIDIRECTIONAL ISOLATORS



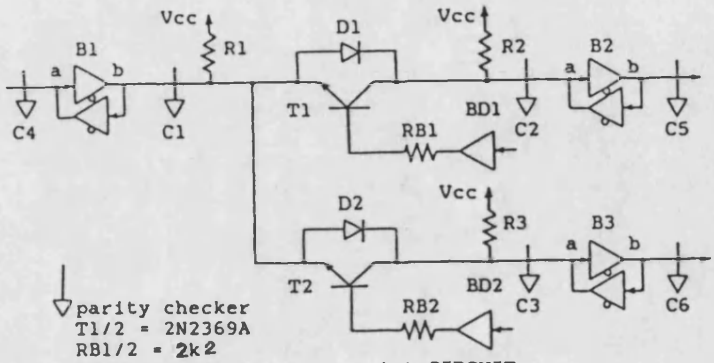
(a) CIRCUIT

FAULT	B1 TX TO B3				B2 TX TO B3			
	C1	C2	C3	C4	C1	C2	C3	C4
B1 O/P SA0	X^		X^	X	X^		X^	X
B1 O/P SA1	X		X	X				
B2 O/P SA0		X^	X^	X		X^	X^	X
B2 O/P SA1						X	X	X
B3 I/P SA0			X^	X			X^	X
B3 I/P SA1	#		#	#		#	#	#
D1 O/C			X	X				
D1 S/C						X		
D2 O/C							X	X
D2 S/C	X							

(b) FAULT TABLE

Notes : 1) B1/B2 enabled when transmission is via that buffer.
2) X = fault indicated.
3) X^ = fault indicated with no buffer enabled.
4) # = indeterminate fault : '0' —|— SA1.
5) R keeps B3 input at 1 when B1 or B2 output a 1.
6) R S/C = B3 I/P SA1
R O/C results in B3 input floating and a possible fault.

FIGURE B.4 DIODES AS UNIDIRECTIONAL ISOLATORS - 2



(a) CIRCUIT

FAULT	B1 TX TO B2						B1 TX TO B3						B2 TX TO B1						B3 TX TO B1					
	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6
B1b SA0	X^	X			X		X^		X			X	X^			X			X^			X		
B1b SA1	X	X			X		X		X			X	#	#		#			#		#	#		
B2a SA0	+^	+^			+		+^	+^	+			+	X^	X^		X			X^	X^		X		
B2a SA1	#	#			#								X	X		X								
B3a SA0	+^	+	+^		+		+^		+^			+	X^		X^	X			X^		X^	X		
B3a SA1							#		#			#							X		X	X		
BD1 SA0		X			X																			
BD1 SA1								X												X				
BD2 SA0									X			X												
BD2 SA1			X												X									
RB1 O/C		X			X																			
RB1 S/C	#	#			#																			
RB2 O/C									X			X												
RB2 S/C							#		#			#												
D1 O/C													X			X				X				
D1 S/C								X																
D2 O/C																			X			X		
D2 S/C			X												X									
T1BE O/C		X			X																			
T1BE S/C		X			X																			
T1BC O/C		X			X																			
T1BC S/C								X																
T1CE O/C		X			X																			
T1CE S/C								X																
T2BE O/C									X			X												
T2BE S/C									X			X												
T2BC O/C									X			X												
T2BC S/C			X												X									
T2CE O/C									X			X												
T2CE S/C			X												X									

(b) FAULT TABLE

Notes : 1) B2/B3 enabled when transmission is via that buffer.
2) X = fault indicated.
3) ^ = fault indicated with no buffer enabled.
4) # = indeterminate fault : '0' —|◁— SA1
5) + = indeterminate fault : SA0 —▷|— '1'
6) Transistor BC S/C provides a diode (BE) as required, except that it is permanently on.
7) R1 S/C ≡ B1b SA1) Any of these resistors open
R2 S/C ≡ B2a SA1 } circuit results in a floating
R3 S/C ≡ B3a SA1) input and a possible fault.

FIGURE B.5 A BIDIRECTIONAL ISOLATING CIRCUIT

NORMAL OPERATION

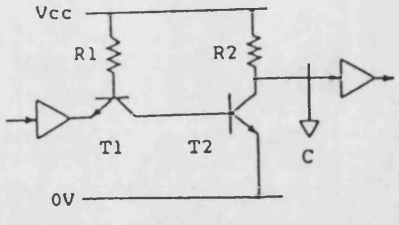
	CHECKER INDICATIONS						SINGLE FAULTS DETECTED
	C1	C2	C3	C4	C5	C6	
^	X	X			X		B1b SA0/1 ($\xrightarrow{2}$),R1 S/C ($\xrightarrow{2}$),B2a SA0 ($\xrightarrow{2}$)
^	X		X			X	B1b SA0/1 ($\xrightarrow{3}$),R1 S/C ($\xrightarrow{3}$),B3a SA0 ($\xrightarrow{3}$)
^	X			X			B1b SA0/1 ($\xleftarrow{2}\xleftarrow{3}$), D1 O/C ($\xleftarrow{2}$), D2 O/C ($\xleftarrow{3}$),R1 S/C ($\xleftarrow{2}\xleftarrow{3}$)
	X	X		X			B2a SA0 ($\xleftarrow{2}\xleftarrow{3}$),B2a SA1 ($\xleftarrow{2}$),R2 S/C ($\xleftarrow{2}$)
	X		X	X			B3a SA0 ($\xleftarrow{2}\xleftarrow{3}$),B3a SA1 ($\xleftarrow{3}$),R3 S/C ($\xleftarrow{3}$)
		X			X		T1BE S/C or O/C ($\xrightarrow{2}$),T1CE O/C ($\xrightarrow{2}$),T1BC O/C ($\xrightarrow{2}$),BD1 SA0 ($\xrightarrow{2}$),RB1 O/C ($\xrightarrow{2}$),B2a SA1 ($\xrightarrow{2}$),R2 S/C ($\xrightarrow{2}$)
		X					T1CE S/C ($\xleftrightarrow{3}$),BD1 SA1 ($\xleftrightarrow{3}$),D1 S/C ($\xleftrightarrow{3}$),T1BC S/C ($\xleftrightarrow{3}$)
			X			X	T2BE S/C or O/C ($\xrightarrow{3}$),T2CE O/C ($\xrightarrow{3}$),T2BC O/C ($\xrightarrow{3}$),BD2 SA0 ($\xrightarrow{3}$),RB2 O/C ($\xrightarrow{3}$),B3a SA1 ($\xrightarrow{3}$),R3 S/C ($\xrightarrow{3}$)
			X				T2CE S/C ($\xleftrightarrow{2}$),BD2 SA1 ($\xleftrightarrow{2}$),D2 S/C ($\xleftrightarrow{2}$),T2BC S/C ($\xleftrightarrow{2}$)
	X	X					B2a SA0 ($\xrightarrow{3}$)
	X		X				B3a SA0 ($\xrightarrow{2}$)

NO BUFFERS ENABLED

X						B1b SA0
X	X					B2a SA0
X		X				B3a SA0

NOTES : 1) X = fault indicated.
2) SA0 = stuck-at-0 ; SA1 = stuck-at-1.
3) O/C = open circuit ; S/C = short circuit.
4) $\xleftarrow{2}$ = read from B2 ; $\xrightarrow{2}$ = write to B2.
5) $\xleftarrow{3}$ = read from B3 ; $\xrightarrow{3}$ = write to B3.
6) For fault diagnostic purposes:
a) R1 associated with B1.
b) D1, T1, RB1, R2 and BD1 associated with B2.
c) D2, T2, RB2, R3 and BD2 associated with B3.
7) RB1 or RB2 short circuit are both undetectable.
8) ^ = fault not diagnosable to B1, B2 or B3 from checker indication.

FIGURE B.6 FAULT ANALYSIS FOR THE BIDIRECTIONAL ISOLATOR



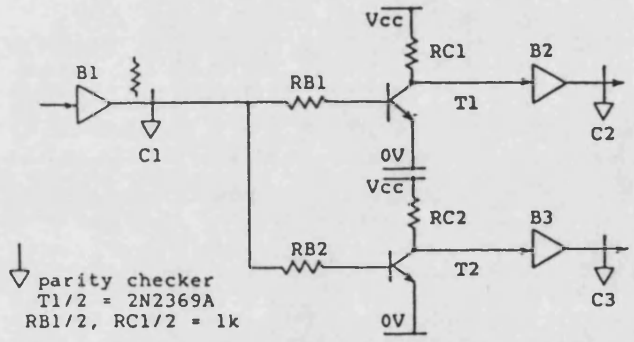
(a) CIRCUIT

T1BE S/C, T1CE S/C and R2 S/C not detected

FAULT	C
T1BE O/C	X
T1BE S/C	
T1BC O/C	X
T1BC S/C	X
T1CE O/C	X
T1CE S/C	
T2BE O/C	X
T2BE S/C	X
T2BC O/C	X
T2BC S/C	X
T2CE O/C	X
T2CE S/C	X
R1 O/C	X
R1 S/C	X
R2 O/C	X
R2 S/C	

(b) FAULT TABLE

FIGURE B.7 UNSWITCHED UNIDIRECTIONAL ISOLATOR - 1



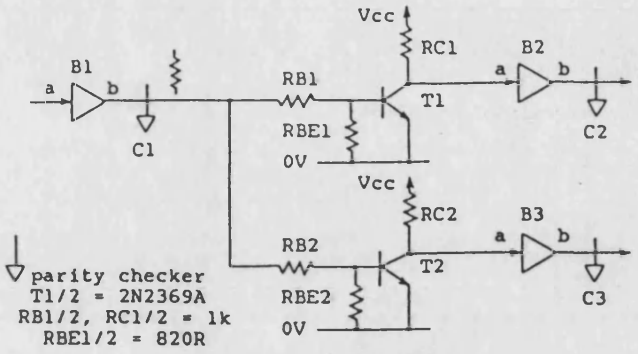
(a) CIRCUIT

FAULT	C1	C2	C3	FAULT	C1	C2	C3
B1 O/P SA0	X	X	X	T2BE S/C			X
B1 O/P SA1	X	X	X	T2BC O/C			X
B2 I/P SA0		X		T2BC S/C			X
B2 I/P SA1		X		T2CE O/C			X
B3 I/P SA0			X	T2CE S/C			X
B3 I/P SA1			X	RB1 O/C		X	
T1BE O/C		X		RB1 S/C			
T1BE S/C		X		RC1 O/C			
T1BC O/C		X		RC1 S/C		X	
T1BC S/C		X		RB2 O/C			
T1CE O/C		X		RB2 S/C			
T1CE S/C		X		RC2 O/C			X
T2BE O/C			X	RC2 S/C			

RB1 S/C, RB2 S/C, RC1 O/C and RC2 O/C not detected

(b) FAULT TABLE

FIGURE B.8 UNSWITCHED UNIDIRECTIONAL ISOLATOR - 2



(a) CIRCUIT

FAULT	C1	C2	C3	FAULT	C1	C2	C3
B1b SA0	X	X	X	T2BC S/C			X
B1b SA1	X	X	X	T2CE O/C			X
B2a SA0		X		T2CE S/C			X
B2a SA1		X		RB1 O/C		X	
B3a SA0			X	RB1 S/C	X		X
B3a SA1			X	RBE1 O/C			
T1BE O/C		X		RBE1 S/C		X	
T1BE S/C		X		RC1 O/C			
T1BC O/C		X		RC1 S/C		X	
T1BC S/C		X		RB2 O/C			X
T1CE O/C		X		RB2 S/C	X	X	
T1CE S/C		X		RBE2 O/C			
T2BE O/C			X	RBE2 S/C			X
T2BE S/C			X	RC2 O/C			
T2BC O/C			X	RC2 S/C			X

RBE1 O/C, RBE2 O/C, RC1 O/C and RC2 O/C not detected

(b) FAULT TABLE

FIGURE B.9 UNSWITCHED UNIDIRECTIONAL ISOLATOR - 3

CHECKER INDICATIONS			SINGLE FAULTS DETECTED
C1	C2	C3	
X	X	X	B1b SA0/1
	X		B2a SA0/1, ALL T1 FAILURES, RB1 O/C, RBE1 S/C, RC1 S/C
		X	B3a SA0/1, ALL T2 FAILURES, RB2 O/C, RBE2 S/C, RC2 S/C
X		X	RB1 S/C
X	X		RB2 S/C

NOTES : 1) X = fault indicated.
2) SA0 = stuck-at-0 ; SA1 = stuck-at-1.
3) O/C = open circuit ; S/C = short circuit.
4) For fault diagnostic purposes:
a) T1, RB1, RBE1 and RC1 are associated with B2.
b) T2, RB2, RBE2 and RC2 are associated with B3.
5) RBE1, RBE2, RC1, or RC2 open circuit are all undetectable

FIGURE B.10 FAULT ANALYSIS FOR THE UNSWITCHED UNIDIRECTIONAL ISOLATOR - 3